

GAN-based Matrix Factorization for Recommender Systems*

Ervin Dervishaj
 ContentWise, Politecnico di Milano
 Milan, Italy
 ervin.dervishaj@mail.polimi.it

Paolo Cremonesi
 Politecnico di Milano
 Milan, Italy
 paolo.cremonesi@polimi.it

ABSTRACT

Proposed in 2014, Generative Adversarial Networks (GAN) initiated a fresh interest in generative modelling. They immediately achieved state-of-the-art in image synthesis, image-to-image translation, text-to-image generation, image inpainting and have been used in sciences ranging from medicine to high-energy particle physics. Despite their popularity and ability to learn arbitrary distributions, GAN have not been widely applied in recommender systems (RS). Moreover, only few of the techniques that have introduced GAN in RS have employed them directly as a collaborative filtering (CF) model.

In this work we propose a new GAN-based approach that learns user and item latent factors in a matrix factorization setting for the generic top-N recommendation problem. Following the vector-wise GAN training approach for RS introduced by CFGAN, we identify 2 unique issues when utilizing GAN for CF. We propose solutions for both of them by using an autoencoder as discriminator and incorporating an additional loss function for the generator. We evaluate our model, GANMF, through well-known datasets in the RS community and show improvements over traditional CF approaches and GAN-based models. Through an ablation study on the components of GANMF we aim to understand the effects of our architectural choices. Finally, we provide a qualitative evaluation of the matrix factorization performance of GANMF.

CCS CONCEPTS

- Information systems → Recommender systems;
- Computing methodologies → Neural networks; Factorization methods;

KEYWORDS

collaborative filtering, matrix factorization, generative adversarial networks, autoencoder, feature matching

1 INTRODUCTION

With the ever-increasing amount of available digital information, recommender systems (RS) are essential tools in filtering out the content presented to users, providing a personalized experience in various domains. The data generated through the interaction of users with RS gave rise to *collaborative filtering* [34] (CF) which utilizes such user-item interactions to create models that can provide high quality recommendations. Within CF, *latent factor* models are a family of mathematical models that project both users and items into a latent space. Matrix factorization (MF) is the most successful latent factor model for CF [16, 25], made famous during the Netflix

Prize challenge. MF expresses the preference of a user on an item as the dot product between their latent factors.

Generative Adversarial Networks (GAN) [18], proposed by Goodfellow et al. in 2014, are generative models that use neural networks to learn arbitrary probability distributions. GAN have shown impressive results in estimating high-dimensional distributions in computer vision [3, 22, 46], natural language processing [26, 45] and various other fields like physics [14, 30] and medicine [36]. They are an active area of research and multiple architectural variants have been proposed like conditional GAN [28], EBGAN [47], InfoGAN [9], etc. Despite their popularity, GAN have not seen wide adoption in RS. The first work to incorporate GAN in the context of RS was IRGAN [41], proposed in 2017, albeit focusing more generally in merging discriminative and generative paradigms in information retrieval (IR). GraphGAN [40] exploits the graph structure of user-item interactions and utilizes GAN to learn embeddings for nodes in this graph. CFGAN [7] identifies a problem with IRGAN’s training and proposes *vector-wise* training for GAN in RS. Other works utilize GAN with additional information beside the user-item interactions [4, 32], to alleviate the sparsity of RS ratings [6, 42] and to model contextual recommendations [2, 27].

Since the adoption of GAN in RS is still in its early phases, we believe there is still room for improvement, especially in using GAN **explicitly as a CF model**. Current GAN approaches like CFGAN attempt to generate the preferences of a specific user on all items by learning from the real preferences of the user. However, RS are characterized by a high number of items and a *single set* of preferences per user, which makes generating user-specific preferences non-trivial. Based on this, our main contributions in this work are:

- We identify the two issues mentioned above when employing GAN for CF. Motivated by them, we derive GANMF, a new conditional GAN-based latent factor model aimed at the generic top-N recommendation problem under implicit feedback.
- We show that GANMF outperforms both traditional and other GAN baselines in two ranking metrics on 3 well-known datasets in RS community.
- We perform an ablation study on the components of GANMF to better understand how they affect the model performance.
- We investigate the MF model learned by GANMF in terms of the number of latent factors and users with fewer interactions.

The rest of this paper is structured as follows. Section 2 provides a short overview of GAN followed by a presentation of other GAN-based RS. In section 3, we motivate our work through the two key issues of GAN in CF. Section 4 details our proposed model and its components. In section 5, we provide the settings of our experiments (section 5.1) and discuss the obtained results (section

*Accepted as a long paper at the 37th ACM/SIGAPP Symposium on Applied Computing (SAC ’22), Recommender Systems track, <https://doi.org/10.1145/3477314.3507099>.

5.2). We present an ablation study in section 5.3 and further examine our model in the context of MF in section 5.4. Finally, we conclude with section 6.

2 PRELIMINARIES

We formally present the generic problem of top-N recommendation. Given a set of users U , a set of items I and users' past feedback on these items, top-N recommendation is the problem of recommending to every user $u \in U$ a subset of items from I that u (has not previously interacted with) is more likely to enjoy. We can arrange the past feedback into a user rating matrix (URM) of shape $|U| \times |I|$. Every cell (u, i) of the URM represents the feedback of user u on item i . In this work we focus only on implicit feedback, given that it is more abundant and easier to secure [25]. In this case, cell (u, i) of URM has a value of 1 if user u has shown interest in item i or 0 otherwise. Each row of the URM represents the historical profile of a user whereas each column the historical profile of an item.

2.1 Generative Adversarial Networks

GAN are part of a class of generative models called implicit density estimating generative models [17]. They do not assume a fixed form of the data distribution but build a model for the distribution from which we can sample. In a GAN, 2 players are pitted against one another in a minimax zero-sum game. One of the players is the **generator** (\mathcal{G}) and the other the **discriminator** (\mathcal{D}). \mathcal{G} , takes as input a noise vector \mathbf{z} sampled from a predefined distribution p_z and outputs a synthetic data point in the real data space. \mathcal{D} on the other hand, takes as input data coming from the real training data and from the generator and is tasked to differentiate the source of its input. The generator is trained so that it can fool the discriminator into classifying the data it generates as real data. In this setup, both the discriminator and generator optimize the same objective function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z} [\log(1 - D(G(\mathbf{z})))] \quad (1)$$

Goodfellow et al. theoretically prove that for the generator to learn the distribution of the real data, the discriminator must be maximally confused about the source of its input.

2.1.1 Conditional GAN (cGAN). This is a GAN variant in which the generator is guided to produce data that belongs to a given class [28]. This is achieved by concatenating the class on which we want to condition the generation process to the input of both discriminator and generator. The objective function in a cGAN is thus changed to:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{x} \sim p_{data}} [\log D(\mathbf{x}|c)] + \mathbb{E}_{\mathbf{z} \sim p_z} [\log(1 - D(G(\mathbf{z}|c)))] \quad (2)$$

2.2 Related Work

IRGAN pioneered GAN in IR and RS. It focuses in providing a unification of generative and discriminative paradigms of modelling in IR. Given a set of queries $\{q_1, \dots, q_N\}$, a set of documents $\{d_1, \dots, d_M\}$ and the true relevancy distribution $p_{true}(d|q, r)$ of documents to queries, IRGAN learns $p_\theta(d|q, r)$ through a generator G . $p_\theta(d|q, r)$ is such that, when sampled from, a binary classifier D cannot distinguish whether the document is coming from p_θ or

from p_{true} . IRGAN optimizes the following minimax function:

$$\min_{\theta} \max_{\phi} \sum_{n=1}^N \left(\mathbb{E}_{d \sim p_{true}(d|q_n, r)} [\log D(d|q_n)] + \mathbb{E}_{d \sim p_\theta(d|q_n, r)} [\log(1 - D(d|q_n))] \right) \quad (3)$$

where the parameters θ of G are updated through the REINFORCE algorithm [43] due to the non-differentiable discrete sampling from p_θ . CFGAN [7], a cGAN approach for RS, takes the training process of IRGAN and experimentally shows that the discrete sampling operation in the generator deteriorates the performance of the discriminator. It proposes *vector-wise* training as a solution where the generator produces full user historical profiles. The discriminator differentiates between generated profiles and real user profiles coming from the URM. CAAE [8] uses 2 autoencoder-based generators as a replacement for MF-based generator in IRGAN and pairs those with BPR [33] loss in the discriminator. To feed the discriminator, one of the generators samples positive items for a given user whereas the other samples negative ones. RAGAN [6] and AugCF [42] focus on alleviating the sparsity of the URM and then apply traditional CF models to provide recommendations. RAGAN utilizes the recommendation capabilities of CFGAN to generate explicit ratings for items and uses them to impute the missing ratings in the URM. RAGAN^{BT} [6] fixes RAGAN's bias towards generating *high-value ratings* by incorporating the application of CDAE [44] on the user historical profile before using CFGAN to fill the URM. AugCF is a cGAN where the generator takes as input a user, unvisited items for this user, associated side information and a class (like or dislike) and outputs a plausible item for the user under the selected class. The discriminator in AugCF takes two roles; in a first phase it acts as a classifier whereas in a second phase it functions as a pure CF model. Other works use GAN with additional information beside the user-item interactions. AugCF can also be considered such a model. [4] tackles the task of learning node representations in a bibliographic network by combining the content of papers with the adjacency matrix between papers and authors.

3 MOTIVATION

We highlight two issues that arise when utilizing a cGAN to **generate plausible user profiles** (as in the case of CFGAN). The discriminator of a cGAN is usually a binary classifier; it outputs a scalar value indicating the probability that the input is coming from the real data. However, the output of the generator in the case of CFGAN is very high dimensional; specifically its dimension is the length of a user historical profile $|I|$, which for some datasets can be in hundreds of thousands or even millions of elements. Updating the weights of the generator through the gradient of a single scalar value in the discriminator output poses difficulties for learning the generator function. As an example, given user u 's real profile I_u^* , we consider the case of 2 generated profiles for user u , \hat{I}_u^1 and \hat{I}_u^2 . \hat{I}_u^1 differs from I_u^* on only some items whereas \hat{I}_u^2 is the inverse of I_u^* . For an optimal binary classifier, both \hat{I}_u^1 and \hat{I}_u^2 are fake and the gradient of the loss propagated back to the generator would be more or less the same for both. However, under some distance metric, we have:

$$\|I_u^* - \hat{I}_u^1\| \leq \|I_u^* - \hat{I}_u^2\| \quad (4)$$

yet the gradient coming from the discriminator to the generator would not make this distinction clear.

A cGAN takes a class label as part of the input in both generator and discriminator so that the generated data are conditioned on the class. Applications of cGAN usually involve datasets with multiple classes where for each class there are hundreds or thousands of samples from the training set. However, in the case of CFGAN where a user identifier is considered the conditioning class, there is only a single profile per user. Any model's ability to learn the input-target function from a single data point per target label is very limited.

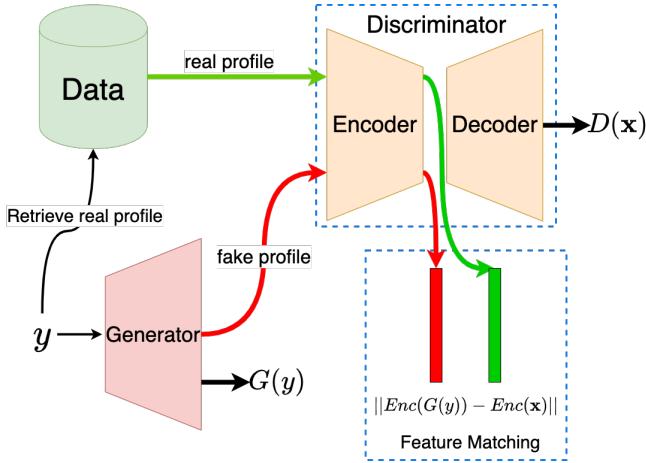


Figure 1: GANMF architecture. The generator is fed a user identifier y and produces a plausible user profile. The discriminator is trained with real profiles retrieved from the URM and with fake profiles coming from the generator.

4 GANMF

We propose a new model, GANMF, that circumvents both issues raised in section 3. GANMF takes the form of a cGAN; the generator is conditioned through a user identifier y to produce a plausible user profile belonging to y , whereas the discriminator is trained to differentiate between profiles produced by the generator and real profiles retrieved from the URM. The optimization of these 2 networks is alternated until the discriminator cannot distinguish the profiles produced by the generator from the real ones. Figure 1 gives a visual depiction of the complete architecture of GANMF.

4.1 GANMF Discriminator

In order to provide richer gradients to the generator, in GANMF we replace the binary classifier discriminator of cGAN with an autoencoder. The autoencoder takes as input a user profile, either from the URM or one generated by the generator, and outputs its reconstruction. EBGAN [47] was first to introduce an autoencoder-based discriminator as an energy function that assigns low energy to data from the training set and high energy to data produced by the generator, thus differentiating the source of the input of the discriminator. In GANMF, the reconstruction error of the autoencoder acts as the energy function:

$$D(x) = \|Dec(Enc(x)) - x\|_2^2 \quad (5)$$

where $Dec(\cdot)$ and $Enc(\cdot)$ are the respective decoder and encoder functions of the autoencoder and $\|\cdot\|_2$ is the Euclidean norm. Given a conditioning user identifier y and its real profile x , the discriminator \mathcal{D} **minimizes** a hinge loss function:

$$\mathcal{L}_{\mathcal{D}}(x, y) = D(x) + [mD(x) - D(G(y))]^+ + \lambda_{\mathcal{D}}\|\Omega_{\mathcal{D}}\|_2^2 \quad (6)$$

where $[\cdot]^+ = \max(0, \cdot)$, $D(\cdot)$ is the discriminator function as defined in (5), $G(\cdot)$ is the generator function, $\lambda_{\mathcal{D}}$ is a regularization coefficient and $\Omega_{\mathcal{D}}$ is the set of parameters of \mathcal{D} . Different from EBGAN, instead of a positive margin we use a positive margin coefficient m in order to limit the range of its values. Optimizing (6) forces the reconstruction error of real profiles towards zero whereas the reconstruction error of generated profiles m -times that of the real ones. Note that different from [37] where the reconstruction loss of the autoencoder is computed only on past true interactions, in GANMF the discriminator loss is computed over all items. This is because the autoencoder is only used to differentiate the source of the profile and not directly deriving recommendations from it. Finally, we point out that GANMF discriminator does not take a user conditioning vector like CFGAN, CAAE and RAGAN do.

4.2 GANMF Generator

Generator \mathcal{G} of GANMF is a conditional generator that takes as input a conditioning attribute y that is unique for each user. Contrary to the original formulation of cGAN, we drop the noise vector z in the input in order to have deterministic mapping from the conditioning attribute to the generated profile.

We cast \mathcal{G} into a MF model by using 2 embedding layers, $\Sigma \in \mathbb{R}^{|U| \times K}$ and $V \in \mathbb{R}^{|I| \times K}$, with Σ and V being the user and item latent factors and K the number of latent features. Our training data is composed of user-item interactions only, so we use as conditioning attribute for the generator the row number of each user in the URM. In the forward pass of the generator, y is utilized to retrieve the y -th row from Σ . A synthetic user profile is produced by (see figure 2):

$$G(y) = V\Sigma[y, :]^\top \quad (7)$$

In order to fool the discriminator, the generator **minimizes** the reconstruction error of the discriminator on generated profiles:

$$\mathcal{L}_{\mathcal{G}}(y) = D(G(y)) + \lambda_{\mathcal{G}}\|\Omega_{\mathcal{G}}\|_2^2 \quad (8)$$

where $D(\cdot)$ and $G(\cdot)$ are the discriminator and generator functions respectively, y is the user row in the URM, $\lambda_{\mathcal{G}}$ is the L_2 regularization coefficient and $\Omega_{\mathcal{G}}$ is the set of parameters of the generator. We note that a clear advantage of using embedding layers is that the number of parameters to be learned by the generator is $\Theta(K(|I| + |U|))$, similar to baselines like WRMF [21]. Moreover, the generator of GANMF is **simpler** than other GAN-based RS approaches in that it models only the linear interactions between user and item latent factors and does not incorporate non-linearities.

As mentioned in section 3, training GANMF with a single real profile per user (the analogy of a single sample per class) causes the generator \mathcal{G} to disregard the conditioning attribute during the generation process and to produce a single profile for every user which reduces substantially the recommendations' quality.

To alleviate this problem we incorporate in $\mathcal{L}_{\mathcal{G}}$ an additional loss term called *feature matching* [35]. Feature matching is presented

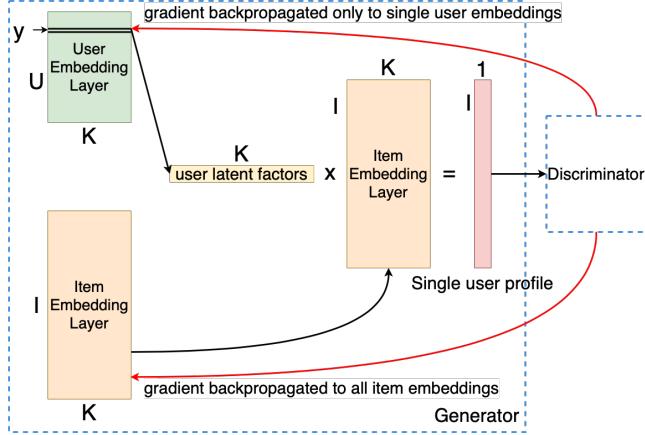


Figure 2: Generator network casted as MF-based approach with embedding layers.

as a solution to stabilize the training procedure of GAN by making the generator produce data that match the statistics of the real data. However, in this work we are interested in \mathcal{G} producing user specific profiles. Feature matching is closely related to *Maximum Mean Discrepancy* [19, 38] which is a distance in probability space between two distributions. Therefore, we modify \mathcal{L}_G by adding this additional loss:

$$\mathcal{L}_G(\mathbf{x}, \mathbf{y}) = (1 - \alpha)D(G(\mathbf{y})) + \alpha \left\| Enc(\mathbf{x}) - Enc(G(\mathbf{y})) \right\|_2^2 + \lambda_G \|\Omega_G\|_2^2 \quad (9)$$

where $Enc(\cdot)$ is the output of the *encoder* in GANMF’s discriminator, \mathbf{y} is the user conditioning attribute, \mathbf{x} is \mathbf{y} ’s real user profile and α is a constant that balances the adversarial and feature matching losses. For GANMF we use a bottlenecked autoencoder which makes the autoencoder learn meaningful features in its *coding layer* [39]. Minimizing \mathcal{L}_G forces generated profiles to match the distribution of real profiles in the latent space induced by the coding layer. This drives \mathcal{G} to generate profiles that cover the same distribution as the real user profiles in this latent space.

5 EXPERIMENTS

5.1 Settings and Evaluation

In our experiments we consider as standard GANMF the model with **single-hidden-layer autoencoder with linear activations** as discriminator and the generator with **2 embedding layers** and **feature matching loss** as described in section 4.2. The final recommendations are given by computing each user’s profile with the generator and then ranking the items the user has not interacted with. Just like CFGAN, our model also has 2 training modes; user-based (*GANMF-u*) and item-based (*GANMF-i*).

We utilize bayesian optimization¹ [1] to find the best hyperparameters for GANMF and the baselines (**all baselines are trained from scratch in our dataset splits**). For each algorithm we optimize MAP@5 on a holdout set and perform 50 runs with the first 10 being random evaluations that seed the Gaussian Process. For

¹We use the Python library scikit-optimize (<https://scikit-optimize.github.io/stable/>) for the bayesian optimization.

Algorithm 1: GANMF Training

```

Input : set of users U, URM, set of parameters  $\Omega_D$ , set of parameters  $\Omega_G$ , margin coefficient m, feature matching coefficient  $\alpha$ , learning rates  $\mu_D$  and  $\mu_G$ , batch size B
Output: trained  $\mathcal{G}$  model that can generate historical user profiles

initialize( $\mathcal{D}, \mathcal{G}$ )
numIterations  $\leftarrow \frac{|U|}{|B|}$ 
while stopping condition not met do
    for iter in numIterations do
        // Discriminator learning
         $y \leftarrow$  sampleBatch(U, B)
        fakeProfiles  $\leftarrow \mathcal{G}(y)$ 
        realProfiles  $\leftarrow$  URM [ $y, :]$ 
         $\mathcal{L}_D \leftarrow$  compute(realProfiles, fakeProfiles)
         $\Omega_D \leftarrow \Omega_D - \mu_D \frac{\partial \mathcal{L}_D}{\partial \Omega_D}$ 
        // Generator learning
         $y \leftarrow$  sampleBatch(U, B)
        fakeProfiles  $\leftarrow \mathcal{G}(y)$ 
         $\mathcal{L}_G \leftarrow D(\text{fakeProfiles})$ 
         $\Omega_G \leftarrow \Omega_G - \mu_G \frac{\partial \mathcal{L}_G}{\partial \Omega_G}$ 
    end
end

```

GANMF² we use Adam optimizer and tune the following intervals for the hyperparameters³:

- Number of epochs: a maximum value of 300.
- Number of latent factors: integer value in [1 – 250].
- Units in the coding layer of AE: integer value in [4 – 1024].
- Batch size: categorical value in [64, 128, 256, 512, 1024].
- Margin coefficient m : integer value in [1 – 10].
- Feature matching coefficient α : real value in [0.01 – 0.5] with a uniform prior distribution.
- \mathcal{D} and \mathcal{G} learning rates: real value in [0.0001 – 0.01] with a log-uniform prior distribution.
- Regularization coefficient λ_D : real value in $[10^{-6} – 10^{-4}]$ with a log-uniform prior distribution.
- Regularization coefficient λ_G : we set this hyperparameter to 0 since \mathcal{G} does not learn directly from the real training data.

The training procedure for GANMF is given by algorithm 1.

5.1.1 Datasets. We evaluate GANMF on three well-known datasets in the RS community; MovieLens 1M [20], MovieLens HetRec [5] and LastFM [5]. MovieLens datasets contain explicit user ratings on movies with every user having rated at least 20 movies. LastFM contains music artist listening information in the form of triples *(user, artist, listeningCount)* where *listeningCount* represents how many times the *user* listened to the *artist*. The statistics of the datasets are given on table 1.

In this work we focus on implicit feedback so we drop the movie ratings in MovieLens datasets and *listeningCount* in LastFM and keep only the interactions between users and items from which we build the URM (we denote this as the *full URM*). We randomly split each dataset into train and test sets in 4:1 ratio. In order to have at least one item per user in both sets we consider only users that have

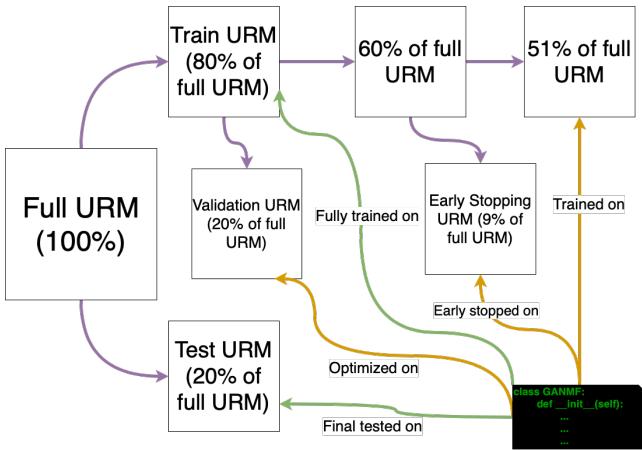
²We implemented GANMF with Tensorflow. Both the generator and the discriminator architectures are highly parallelizable through a GPU which helps limit training/inference times.

³Epochs and number of latent factors hyperparameters are shared by all baselines. The same interval is used in all models.

Table 1: Dataset statistics.

Dataset	Interactions	Users	Items	Sparsity
ML 1M	1000209	6040	3706	95.53%
ML HetRec	855598	2113	10109	96.00%
LastFM	92834	1884	17626	99.72%

interacted with at least 2 items. We use the test set **only** for the final evaluation of all algorithms. In order to tune hyperparameters we further split the training set to obtain validation and early stopping sets. Once we have the dataset-specific hyperparameters, we fully train each algorithm on the initial train set. In figure 3 we give a summary of the steps from the full URM to each of the sets and how they are used by the algorithms.

**Figure 3: Splitting of the full URM into subsequent sets for each of the operations performed by the algorithms.**

5.1.2 Evaluation. We compare all algorithms on the test set through 2 different metrics, *normalized discounted cumulative gain* (NDCG) [23] and *mean average precision* (MAP) on recommendation lists of 5 and 20 items. Both of these metrics take into consideration the ranking of recommended items beside their relevancy. The different cutoffs give us an indication how the algorithms behave with increasing recommendation list length.

The code for all algorithms⁴, along with the datasets' splits, experiments and results can be found in <https://github.com/edervishaj/GANMF>.

5.2 Results

We compare our proposed model with 8 other baseline models aimed at the top-N recommendation problem, including robust baselines [12, 13]:

- **Top-popular:** non-personalized approach where the most popular items are recommended to every user.

⁴For the baselines and model evaluation we use implementations from https://github.com/MaurizioFD/RecSys_Course_AT_PoliMi

- **PureSVD** [11]: MF approach that utilizes SVD to reconstruct the URM. We tune only the number of latent factors for this model.
- **WRMF** [21]: MF technique that converts implicit feedback into confidence values and employs *alternating least squares* for the computation of user and item latent factors. We tune all the parameters as given in the paper.
- **ItemKNN** [15]: one of the main model-based CF techniques that builds an item-item similarity matrix from the URM. For ItemKNN we tune the similarity⁵, the neighborhood size and the shrink term.
- **P³α** [10]: a graph-based recommendation approach where the similarity between items is expressed as a 2-step random walk starting from users in the bipartite graph of users and items. For this model we tune the neighborhood size and similarity scaling coefficient α .
- **SLIM** [29]: a machine learning technique that models the item-item similarity matrix and is trained with BPR [33] loss. For SLIM we tune the hyperparameters as given in the paper.
- **CFGAN** [7]: GAN-based RS that proposes *vector-wise* training for GAN in RS. We tune all the hyperparameters in the ranges provided by the authors.
- **CAAE** [8]: GAN-based RS that incorporates BPR loss in the discriminator and uses 2 autoencoder-based generators. For CAAE we tune the hyperparameters in the ranges provided by the authors.

For CFGAN and GANMF we give both user and item-based variants. For each of the baselines we perform hyperparameter tuning as explained in section 5.1. Table 2 summarizes the comparison between GANMF and all baselines.

We note that we have omitted some GAN-based works. We have omitted IRGAN since CFGAN shows a clear improvement over it. RAGAN^{BT} is intended to work with explicit ratings instead of implicit feedback. Moreover, both RAGAN^{BT} and AugCF use GAN to alleviate sparsity of URM and still rely entirely on traditional CF models to provide the recommendations. [4] and AugCF also rely on side information for their recommendations while we focus only on the user-item interactions.

5.2.1 Discussion. We provide here a brief breakdown of the obtained results in table 2. GANMF variants show superior performance against all baselines in the 3 datasets. In particular, GANMF performs on average 24% better than CFGAN models despite using the same *vector-wise* training procedure. We attribute this disparity to the autoencoder discriminator which is especially helpful when the generator is tasked to generate high dimensional vectors as in the case of RS. Additionally, between the two GANMF variants there is little difference which we were not expecting, especially for LastFM and MovieLens HetRec datasets where the ratio number of users to number of items is higher. On the other hand, CFGAN variants show the opposite behavior, with the user variant consistently surpassing the item one. For the other GAN-based algorithm, CAAE, we were not able to find an official implementation so we tried to implement it following only the paper. However, we found CAAE's recommendation quality to not match the one reported in

⁵Considered similarities: dice, jaccard, tversky, asymmetric-cosine, cosine. We report only ItemKNN-*cossine* since it performed best.

Table 2: Experimental results of GANMF and chosen baselines over MovieLens 1M, MovieLens HetRec and LastFM. The best results per dataset and per metric are given in bold, second best results are underlined.

Algorithm	ML 1M				ML HetRec				LastFM			
	NDCG @5	MAP @5	NDCG @20	MAP @20	NDCG @5	MAP @5	NDCG @20	MAP @20	NDCG @5	MAP @5	NDCG @20	MAP @20
Top Popular	0.2248	0.1544	0.1952	0.0919	0.4770	0.3899	0.3905	0.2475	0.0888	0.0524	0.0947	0.0392
PureSVD	0.4197	0.3243	0.3644	0.2139	0.5933	0.5126	0.5020	0.3604	0.2263	0.1505	0.2145	0.1064
WRMF	0.4229	0.3200	0.3783	0.2178	0.5762	0.4859	0.4923	0.3393	0.2745	0.1848	0.2623	0.1336
$P^3\alpha$	0.4066	0.3086	0.3553	0.2016	0.5432	0.4539	0.4532	0.3028	0.2469	0.1635	0.2370	0.1154
ItemKNN-cos	0.4088	0.3121	0.3577	0.2063	0.5599	0.4783	0.4664	0.3215	0.2683	0.1804	0.2566	0.1277
SLIM	0.4298	0.3249	0.3775	0.2147	0.5643	0.4710	0.4862	0.3284	0.2172	0.1343	0.2223	0.1008
CAAE	0.2217	0.1531	0.1941	0.0913	0.4767	0.3894	0.3902	0.2467	0.0834	0.0504	0.0940	0.0378
CFGAN-u	0.4044	0.3066	0.3487	0.1977	0.5123	0.4151	0.4224	0.2695	0.2358	0.1482	0.2338	0.1079
CFGAN-i	0.2211	0.1546	0.1909	0.0928	0.4462	0.3533	0.3707	0.2267	0.2219	0.1433	0.2145	0.1021
GANMF-u	0.4564	0.3551	0.4032	0.2423	<u>0.6076</u>	<u>0.5255</u>	<u>0.5151</u>	<u>0.3715</u>	<u>0.2857</u>	<u>0.1936</u>	0.2742	0.1402
GANMF-i	<u>0.4500</u>	<u>0.3503</u>	<u>0.3985</u>	<u>0.2399</u>	0.6230	0.5445	0.5276	0.3866	0.2865	0.1943	<u>0.2725</u>	<u>0.1397</u>

the original paper. In our dataset splits it performed on par with the non-personalized technique.

The two latent factor models, PureSVD and WRMF have comparable performance between them in all datasets. The best performing GANMF variant is on average 15% and 8% better than PureSVD and WRMF, respectively, in the MovieLens datasets. In LastFM dataset, GANMF performs on average 29% better than PureSVD. It is clear that the matrix factorization model learned by GANMF is able to find latent factors that explain better the interactions of users and items (we explore this in more details in section 5.4). ItemKNN, SLIM and $P^3\alpha$ all model the item-item similarity matrix and as such, their performances do not differ much. They also tend to score slightly lower than MF techniques. Compared to GANMF, the best neighborhood model is 10% worse across metrics and datasets. We conclude that GANMF is able to outperform baselines in GAN-based models and traditional approaches.

5.3 Ablation Study

In order to study the components of GANMF, we perform 2 different experiments where we replace one component at a time.

5.3.1 GANMF with binary classifier discriminator. In this experiment we drop the autoencoder discriminator in GANMF and replace it with a binary classifier discriminator just like in vanilla cGAN and CFGAN. We denote this model *binGANMF*. This new discriminator outputs the probability of its input coming from the URM. In order to evaluate only the impact of the discriminator, we keep everything else exactly as explained in section 4; generator with embedding layers and feature matching. As user features to match, we use the learned features in the last layer of the discriminator before the final output. We retrain again binGANMF with bayesian optimization. On table 3 we report its performance on the test set along with the standard GANMF. We see that when equipped with an autoencoder discriminator, GANMF variants performs much better than binGANMF on all metrics, as much as 4 times better on LastFM dataset on MAP@20 metric.

5.3.2 Effect of Feature Matching. In order to understand how feature matching affects GANMF, we modify only the feature matching coefficient α in equation 9 in the range [0 – 1] with a step of 0.2 and rerun bayesian optimization again for the resulting GANMF variants. In figure 4 we show how different values of α change the performance of GANMF for MovieLens 1M, MovieLens HetRec and LastFM, respectively (for space reasons we give only GANMF-i, similar behavior is observed for GANMF-u). On all datasets and the 2 different cutoffs, a combination of both the adversarial GAN loss and feature matching provides the best results for GANMF.

The other important aspect of using feature matching is to enforce conditional generation for the generator. To investigate its usefulness, we train GANMF with and without feature matching and after each training phase, we produce profiles for all users. Then, we compute the cosine similarity between each pair of users and show them as heatmaps in figure 5. We observe significant decrease in user-user similarity after the application of feature matching. This means that the generator outputs less similar vectors and each user conditioning attribute is mapped to a more unique profile.

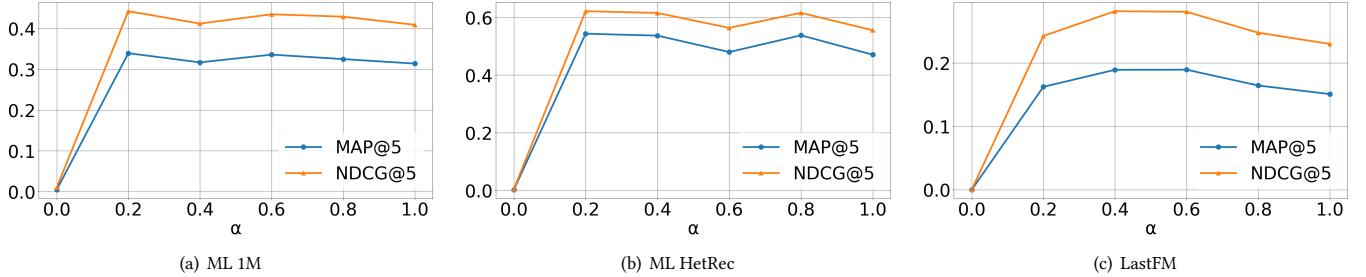
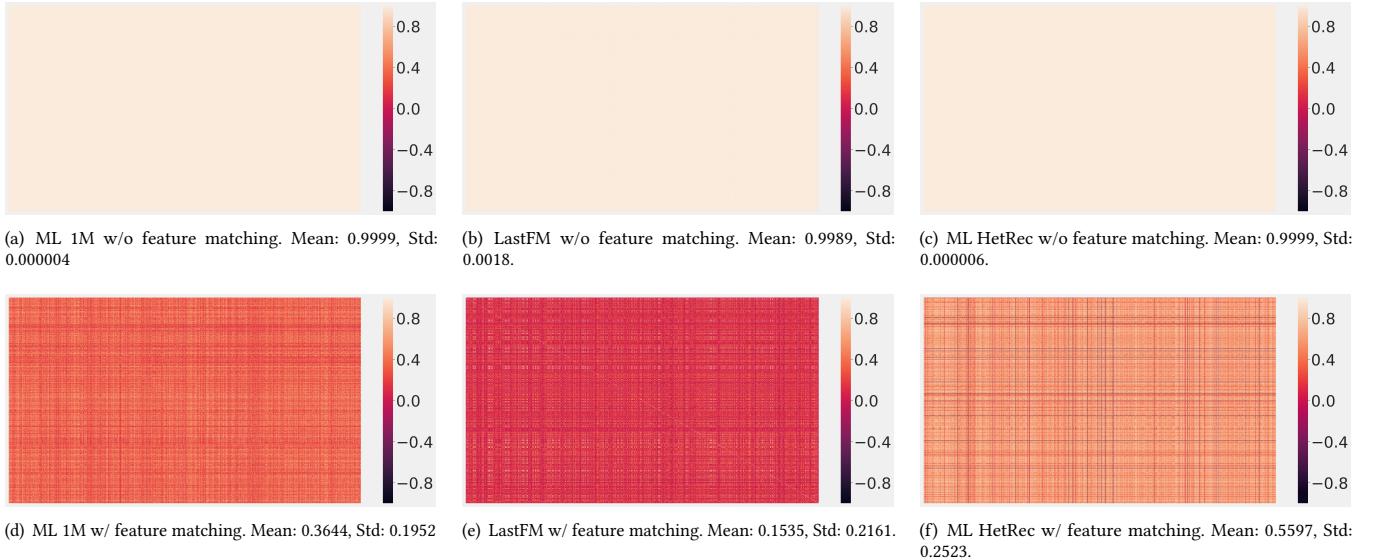
5.4 On the MF model learned by GANMF

Given the quantitative advantage of GANMF over the considered baselines, it is important to investigate the latent factors model learned by GANMF.

We compare the behavior of GANMF with varying number of latent factors K along with two other MF baselines, PureSVD and WRMF. For each model we fixed K and tuned the other hyper-parameters following the procedure detailed in section 5.1 with 35 runs instead of 50. Their performance on MAP@5 is given in figure 6. GANMF dominates the other baselines in most of the considered latent factors, with the exception of $K < 100$ on LastFM where WRMF performs better. The best models for PureSVD and WRMF tend to provide more accurate recommendations when using a relatively low number of latent factors since such traditional MF techniques are known to overfit with large K [24]. GANMF on the other hand still improves its recommendation accuracy with increasing K . This is mainly due to GANMF using the discriminator

Table 3: Ablation study. The best results per dataset and per metric are given in bold, the second best results are underlined.

Algorithm	ML 1M				ML HetRec				LastFM			
	NDCG @5	MAP @5	NDCG @20	MAP @20	NDCG @5	MAP @5	NDCG @20	MAP @20	NDCG @5	MAP @5	NDCG @20	MAP @20
GANMF-u	0.4564	0.3551	0.4032	0.2423	<u>0.6076</u>	0.5255	<u>0.5151</u>	0.3715	<u>0.2857</u>	0.1936	0.2742	0.1402
GANMF-i	<u>0.4500</u>	<u>0.3503</u>	<u>0.3985</u>	<u>0.2399</u>	0.6230	0.5445	0.5276	0.3866	0.2865	0.1943	<u>0.2725</u>	<u>0.1397</u>
binGANMF-u	0.2185	0.1480	0.1920	0.0895	0.4660	0.3776	0.3861	0.2426	0.0790	0.0454	0.0842	0.0337
binGANMF-i	0.2889	0.2075	0.2446	0.1240	0.4707	0.3827	0.3871	0.2440	0.0855	0.0477	0.0966	0.0374

**Figure 4: Effect of feature matching loss on GANMF-i performance on MovieLens 1M, MovieLens HetRec and LastFM.****Figure 5: Feature matching conditioning on the user generated profiles by GANMF-u. The heatmaps represent the user-user similarity with and without feature matching. Mean and standard deviation of the similarities are given for each dataset.**

to learn the latent factors instead of learning them directly from the training data. This is further reinforced by the fact that we do not place a regularization term on the parameters of \mathcal{G} .

Another important aspect is the behavior of GANMF on the different types of users based on the number of items they have interacted with. Figure 7 shows the performance of PureSVD, WRMF and GANMF for 4 types of users. MF models are known to be susceptible to the *cold start* problem with performance suffering for

users that have interacted with few items [31]. This is evident especially on MovieLens 1M and MovieLens HetRec where ranking accuracy is much lower for users that have interacted with less than 25 items and less than 100 items. Even for these users GANMF is able to provide better recommendations than the MF baselines. This can be attributed to the autoencoder discriminator and feature matching loss. In its coding layer, the discriminator \mathcal{D} learns a meaningful representations of users' preference over the items.

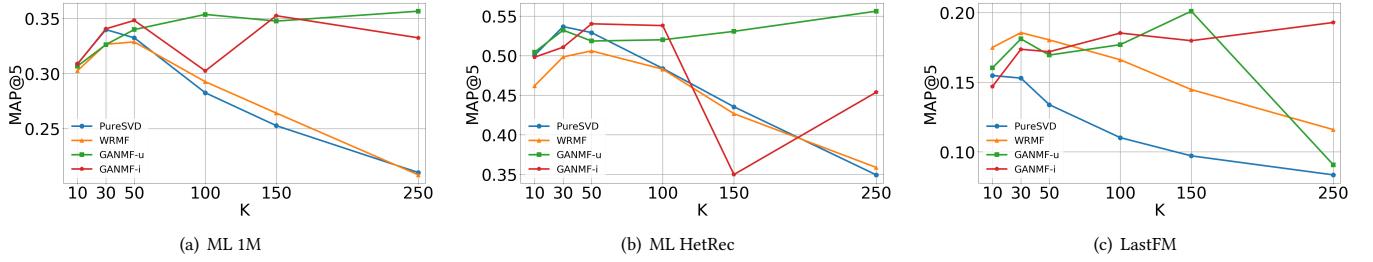


Figure 6: MAP@5 of PureSVD, WRMF and GANMF on MovieLens 1M, MovieLens HetRec and LastFM for varying latent factors number.

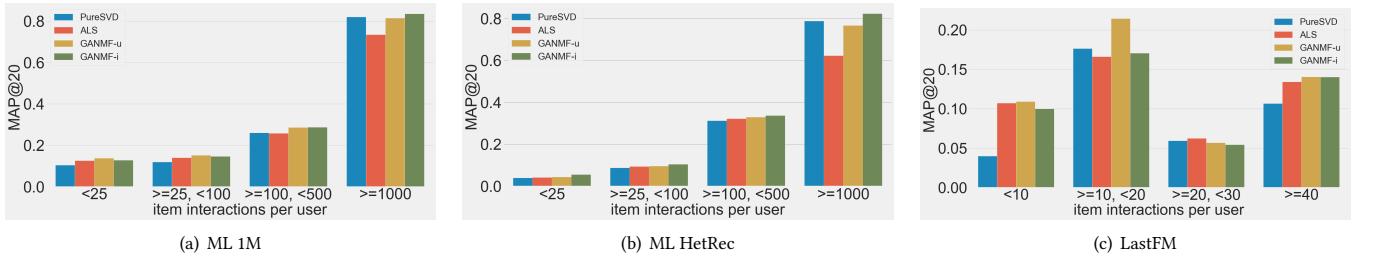


Figure 7: MAP@20 of PureSVD, WRMF and GANMF on MovieLens 1M, MovieLens HetRec and LastFM for varying latent factors number.

Thus, even users that do not share exactly the same items in data space, might share similar latent representation in the coding layer. When trained with feature matching, the generator can transfer some knowledge from the shared latent space back to the latent factors of users with few interactions.

6 CONCLUSION

In this work we presented GANMF, a novel approach to building GAN-based latent factors models for top-N recommendation with implicit feedback. We identified 2 main issues when using cGAN in CF; using a single-output binary-classifier discriminator does not provide rich enough gradients to the generator given the dimensionality of the user/item profiles in RS and the lacking of multiple data samples per user which causes the generator of a cGAN to disregard the conditioning attribute in the case of CF. We give solutions for both of these issues by replacing the binary-classifier discriminator in the original formulation of cGAN with an autoencoder and by incorporating a feature matching loss in the generator. Through an ablation study we show that our model can achieve its best performance with a combination of both GAN loss and feature matching loss. More importantly, we show that feature matching enforces conditional generation of user/item profiles. We tested our proposed model in 3 publicly available datasets of different sparsity and shapes and provided a comparison with 8 other baselines, representatives for latent factor, neighborhood and GAN-based models. We showed that GANMF outperformed all of them in 2 ranking metrics across 3 datasets. Finally, the qualitative results we provided on the latent factors model learned by GANMF indicate its efficiency for users with few interactions. These results show

that GAN are a promising technique that can be used as a main component in RS.

REFERENCES

- [1] Sebastiano Antenucci, Simone Boglio, Emanuele Chiosi, Ervin Dervishaj, Shuwen Kang, Tommaso Scarlatti, and Maurizio Ferrari Dacrema. 2018. Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario. In *Proceedings of the ACM Recommender Systems Challenge 2018*. 1–6.
- [2] Homanga Bharadhwaj, Homin Park, and Brian Y Lim. 2018. RecGAN: recurrent generative adversarial networks for recommendation systems. In *Proceedings of the 12th ACM Conference on Recommender Systems*. 372–376.
- [3] Andrew Brock, Jeff Donahue, and Karen Simonyan. 2018. Large scale GAN training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096* (2018).
- [4] Xiaoyan Cai, Junwei Han, and Libin Yang. 2018. Generative adversarial network based heterogeneous bibliographic network representation for personalized citation recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
- [5] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. 2011. Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011). In *Proceedings of the fifth ACM conference on Recommender systems*. 387–388.
- [6] Dong-Kyu Chae, Jin-Soo Kang, Sang-Wook Kim, and Jaeho Choi. 2019. Rating augmentation with generative adversarial networks towards accurate collaborative filtering. In *The World Wide Web Conference*. 2616–2622.
- [7] Dong-Kyu Chae, Jin-Soo Kang, Sang-Wook Kim, and Jung-Tae Lee. 2018. Cfgan: A generic collaborative filtering framework based on generative adversarial networks. In *Proceedings of the 27th ACM international conference on information and knowledge management*. 137–146.
- [8] Dong-Kyu Chae, Jung Ah Shin, and Sang-Wook Kim. 2019. Collaborative adversarial autoencoders: An effective collaborative filtering model under the GAN framework. *IEEE Access* 7 (2019), 37650–37663.
- [9] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657* (2016).
- [10] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. 2014. Random walks in recommender systems: exact computation and simulations. In *Proceedings of the 23rd international conference on world wide web*. 811–816.

GAN-based Matrix Factorization for Recommender Systems

- [11] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*. 39–46.
- [12] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. 2021. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems (TOIS)* 39, 2 (2021), 1–49.
- [13] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 101–109.
- [14] Luke de Oliveira, Michela Paganini, and Benjamin Nachman. 2017. Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. *Computing and Software for Big Science* 1, 1 (2017), 1–24.
- [15] Mukund Deshpande and George Karypis. 2004. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 143–177.
- [16] Simon Funk. 2006. Netflix update: Try this at home.
- [17] Ian Goodfellow. 2016. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160* (2016).
- [18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
- [19] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *The Journal of Machine Learning Research* 13, 1 (2012), 723–773.
- [20] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (itis)* 5, 4 (2015), 1–19.
- [21] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 263–272.
- [22] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1125–1134.
- [23] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
- [24] Christopher C Johnson. 2014. Logistic matrix factorization for implicit feedback data. *Advances in Neural Information Processing Systems* 27, 78 (2014), 1–9.
- [25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
- [26] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. 2017. Adversarial ranking for language generation. *arXiv preprint arXiv:1705.11001* (2017).
- [27] Wei Liu, Zhi-Jie Wang, Bin Yao, and Jian Yin. 2019. Geo-ALM: POI Recommendation by Fusing Geographical Information and Adversarial Learning Mechanism. In *IJCAI*, Vol. 7. 1807–1813.
- [28] Mehdi Mirza and Simon Osindero. 2014. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).
- [29] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*. IEEE, 497–506.
- [30] Michela Paganini, Luke de Oliveira, and Benjamin Nachman. 2018. CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. *Physical Review D* 97, 1 (2018), 014021.
- [31] Seung-Taek Park and Wei Chu. 2009. Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems*. 21–28.
- [32] Ruiyang Ren, Zhaoyang Liu, Yaliang Li, Wayne Xin Zhao, Hui Wang, Bolin Ding, and Ji-Rong Wen. 2020. Sequential recommendation with self-attentive multi-adversarial network. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 89–98.
- [33] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).
- [34] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.
- [35] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*. 2234–2242.
- [36] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Georg Langs, and Ursula Schmidt-Erfurth. 2019. f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks. *Medical image analysis* 54 (2019), 30–44.
- [37] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*. 111–112.
- [38] Michael Tschannen, Olivier Bachem, and Mario Lucic. 2018. Recent advances in autoencoder-based representation learning. *arXiv preprint arXiv:1812.05069* (2018).
- [39] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research* 11, 12 (2010).
- [40] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. Graphgan: Graph representation learning with generative adversarial nets. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
- [41] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 515–524.
- [42] Qinyong Wang, Hongzhi Yin, Hao Wang, Quoc Viet Hung Nguyen, Zi Huang, and Lizhen Cui. 2019. Enhancing collaborative filtering with generative augmentation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 548–556.
- [43] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
- [44] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining*. 153–162.
- [45] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.
- [46] Han Zhang, Tao Xu, Hongsheng Li, Shaotong Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. 2017. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. 5907–5915.
- [47] Junbo Zhao, Michael Mathieu, and Yann LeCun. 2016. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126* (2016).