PLASTIC: Prioritize Long and Short-term Information in Top-n Recommendation using Adversarial Training

Wei Zhao¹,², Benyou Wang², Jianbo Ye³, Yongqiang Gao², Min Yang¹*, Xiaojun Chen⁴
¹ Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
² Tencent Inc.
³ Pennsylvania State University
⁴ Shenzhen University

Abstract

Recommender systems provide users with ranked lists of items based on individual’s preferences and constraints. Two types of models are commonly used to generate ranking results: long-term models and session-based models. While long-term models represent the interactions between users and items that are supposed to change slowly across time, session-based models encode the information of users’ interests and changing dynamics of items’ attributes in short terms. In this paper, we propose a PLASTIC model, Prioritizing Long And Short-Term Information in top-n reCommendation using adversarial training. In the adversarial process, we train a generator as an agent of reinforcement learning which recommends the next item to a user sequentially. We also train a discriminator which attempts to distinguish the generated list of items from the real list recorded. Extensive experiments show that our model exhibits significantly better performances on two widely used datasets.¹

1 Introduction

With the sheer volume of online information, much attention has been given to data-driven recommender systems. Numerous recommendation techniques have been developed. Three main categories of them are collaborative filtering methods, content-based methods and hybrid methods [Bobadilla et al., 2013; Lu et al., 2015]. In this paper, we aim to develop a method producing a ranked list of n items to a user by collaborative filtering that exploits historical user-item interactions.

Matrix factorization (MF) [Koren et al., 2009] is one of the most successful techniques in the practice of recommendation due to its simplicity, attractive accuracy and scalability. It models the user preference matrix approximately as a product of two lower-rank latent feature matrices representing user profiles and item profiles respectively. Matrix factorization is robust to the data sparsity and cold-start issues in the recommender system.

Despite the appeal of matrix factorization, this technique does not explicitly consider the temporal variability of data [Wu et al., 2017]. Firstly, the popularity of an item may change over time. For example, movie popularity booms or fades, which can be triggered by external events such as the appearance of an actor in a new movie. Secondly, users may change their interests and baseline ratings over time. For instance, a user who tended to rate an average movie as “4 stars”, may now rate such a movie as “3 stars”. Recently, recurrent neural network (RNN) [Hochreiter and Schmidhuber, 1997] has gained significant attention by considering such temporal dynamics for both users and items, and achieved high recommendation quality [Wu et al., 2016b; 2017].

More recent work [Wu et al., 2017] reveals that matrix factorization based and RNN based recommendation approaches are complementary to each other. Specifically, the matrix factorization recommendation approaches make item predictions based on users’ long-term interests which change very slowly with respect to time. On the contrary, the RNN recommendation approaches predict which item will the user consume next, respecting the dynamics of users’ behaviors and items’ attributes in the short term. It therefore motivates us to devise a joint approach that takes advantage of both matrix factorization and RNN, exploiting both long-term and short-term associations among users and items.

In this paper, we propose a novel PLASTIC model, which Prioritizes Long And Short-Term Information in top-n reCommendation using adversarial training. The PLASTIC model employs an adversarial framework to combine the MF and RNN based models for the top-n recommendation, taking the best of each to improve the final recommending performance. In the adversarial process, we simultaneously train two models: a generative model G and a discriminative model D. In particular, the generator G takes the user u and time t as input, and predicts the recommendation list for user i at time t based on the historical user-item interactions. We implement the discriminator D via a Siamese Network that incorporates long-term and session-based ranking model in a pair-wise scenario. The two point-wise networks of Siamese Network share the same set of parameters. The generator G and the discriminator D are optimized with a minimax two-
player game. The discriminator $D$ tries to distinguish the real high-rated items in the training data from the recommendation list generated by the generator $G$, while the training procedure of generator $G$ is to maximize the probability of $D$ making a mistake. Thus, this adversarial process can eventually adjust $G$ to generate plausible and high-quality recommendation list.

We summarize our main contributions as follows:

• To the best of our knowledge, we are the first to use GAN framework to leverage the MF and RNN approaches for top-$n$ recommendation. This joint model adaptively adjusts how the contributions of the long-term and short-term information of users and items are mixed together.

• We propose four hard and soft mixture mechanisms to integrate MF and RNN. We use the hard mechanism to calculate the mixing score straightforwardly and explore several soft mechanisms to learn the temporal dynamics with the help of the long-term profiles.

• Our model uses reinforcement learning to optimize the generator $G$ for generating highly rewarded recommendation list. Thus, it effectively bypasses the non-differentiable task metric issue by directly performing policy gradient update.

• The experimental results show that our model consistently outperforms the state-of-the-art methods.

2 Related Work

Recommender system is an active research field. The authors of [Bobadilla et al., 2013; Lu et al., 2015] describe most of the existing techniques for recommender systems.

Matrix factorization for recommendation Modeling the long-term interests of users, the matrix factorization method and its variants have grown to become dominant in the literature [Rennie and Srebro, 2005; Koren et al., 2009; Hernando et al., 2016; He et al., 2016; Tu et al., 2015]. In the standard matrix factorization, the recommendation task can be formulated as inferring missing values of a partially observed user-item matrix [Koren et al., 2009]. Srebro et al. [2005] suggested the Maximum Margin Matrix Factorization (MMMF), which used low-norm instead of low-rank factorizations. Mnih and Salakhutdinov [2008] presented the Probabilistic Matrix Factorization (PMF) model that characterized the user preference matrix as a product of two lower-rank user and item matrices. The PMF model was especially effective at making better predictions for users with few ratings. He et al. [2016] proposed a new MF method which considers the implicit feedback for on-line recommendation.

Recurrent neural network for recommendation These traditional MF methods for recommendation systems are based on the assumption that the user interests and item attributes are near static, which is however not consistent with reality. Koren [2010] discussed the effect of temporal dynamics in recommender systems and proposed a temporal extension of the SVD++ (called TimeSVD++) to explicitly model the temporal bias in data. However, the features used in TimeSVD++ were hand-crafted and computationally expensive to obtain. Recently, there have been increasing interests in employing recurrent neural network to model temporal dynamic in recommendation systems. For example, Hidasi et al. [2015] applied recurrent neural network (GRU) to session-based recommender systems. Wu et al. [2016a] proposed a recurrent neural network to perform the time heterogeneous feedback recommendation. Wu et al. [2017] used LSTM autoressive model for the user and item dynamics and employed matrix factorization to model the stationary components that encode fixed properties. Different from their work, we use GAN framework to leverage the MF and RNN approaches for top-$n$ recommendation, aiming to generate plausible and high-quality recommendation lists.

Generative adversarial network for recommendation In parallel, previous work has demonstrated the effectiveness of generative adversarial network (GAN) [Goodfellow et al., 2014] in various tasks such as image generation [Reed et al., 2016], image captioning [Chen et al., 2017], and sequence generation[Yu et al., 2017; Liu et al., 2018]. The most related work to ours is [Wang et al., 2017], which proposed a novel IRGAN mechanism to iteratively optimize a generative retrieval component and a discriminative retrieval component. Our approach differs from theirs in several aspects. First, we combine the MF approach and the RNN approach with GAN, exploiting the performance contributions of both approaches. Second, IRGAN does not attempt to estimate the future behavior. In fact, they use future trajectories to infer the historical records, which seems not useful in real-life applications.

3 Our Model

Suppose there is a sparse user-item rating matrix $R$ that consists of $U$ users and $M$ items. Each entry $r_{ij,t}$ denotes the rating of user $i$ on item $j$ at time step $t$. The rating is represented by numerical values from $1$ to $5$, where the higher value indicates the stronger preference. Instead of predicting the rating of a specific user-item pair as is done in [McNee et al., 2006], our model aims to provide users with ranked lists of items (top-$n$ recommendation) [Liu and Yang, 2008].

The PLASTIC model employs an adversarial framework to combine the MF and RNN based models for the top-$n$ item recommendation. In the adversarial process, we simultaneously train two models: a generative model $G$ and a discriminative model $D$.

3.1 Matrix Factorization (MF)

The MF framework [Mnih and Salakhutdinov, 2008] models the long-term states (global information) for both users ($e^u$) and items ($e^m$). In its standard setting, the recommendation task can be formulated as inferring missing values of a partially observed user-item rating matrix $R$. The formulation of MF is given by:

$$\arg\min_{e^u,e^m} \sum_{i,j} I_{ij} \left( r_{ij} - \rho \left( (e^u_i)^T (e^m_j) \right) \right)^2 + \lambda^u \|e^u\|_F^2 + \lambda^m \|e^m\|_F^2$$

where $e^u_i$ and $e^m_j$ represent the user and item latent factors in the shared $d$-dimension space respectively. $r_{ij}$ denotes the user $i$’s rating on item $j$. $I_{ij}$ is an indicator function and
3.2 Recurrent Neural Network (RNN)

The RNN based recommender system focuses on modeling session-based trajectories instead of global (long-term) information [Wu et al., 2017]. It predicts future behaviors and provides users with a ranking list given the users’ past history. The main purpose of using RNN is to capture time-varying state for both users and items. Particularly, we use LSTM as the basic RNN unit. Each LSTM unit at time $t$ consists of a memory cell $c_t$, an input gate $i_t$, a forget gate $f_t$, and an output gate $o_t$. These gates are computed from previous hidden state $h_{t-1}$ and the current input $x_t$:

$$[f_t, i_t, o_t] = \text{sigmoid}(W[h_{t-1}, x_t]) \quad (2)$$

The memory cell $c_t$ is updated by partially forgetting the existing memory and adding a new memory content $i_t$:

$$I_t = \tanh(V[h_{t-1}, x_t]) \quad (3)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot I_t \quad (4)$$

Once the memory content of the LSTM unit is updated, the hidden state at time step $t$ is given by:

$$h_t = o_t \odot \tanh(c_t) \quad (5)$$

For simplicity of notation, the update of the hidden states of LSTM at time step $t$ is denoted as $h_t = \text{LSTM}(h_{t-1}, x_t)$.

Here, we use $z_{it}^u \in \mathbb{R}^M$ and $z_{jt}^m \in \mathbb{R}^L$ to represent the rating vector of user $i$ and item $j$ given time $t$ respectively. Both $z_{it}^u$ and $z_{jt}^m$ serve as the input to the LSTM layer at time $t$ to infer the new states of the user and the item:

$$h_{t}^u = \text{LSTM}(h_{t-1}^u, z_{it}^u) \quad (6)$$
$$h_{t}^m = \text{LSTM}(h_{t-1}^m, z_{jt}^m) \quad (7)$$

Here, $h_{t}^u$ and $h_{t}^m$ denote hidden states for user $i$ and item $j$ at time step $t$ respectively.

3.3 RNN and MF Hybrid

The session-based model deals with temporal dynamics of the user and item states, we further incorporate the long-term preference of users and the fixed properties of items. To exploit their advantages together, similar to [Wu et al., 2017], we define the rating prediction function as:

$$r_{ij,t} = g(e_{ij}^u, e_{ij}^m, h_{t}^u, h_{t}^m) \quad (8)$$

where $g(\cdot)$ is a score function, $e_{ij}^u$ and $e_{ij}^m$ denote the global latent factors of user $i$ and item $j$ learned by Eq. (1); $h_{t}^u$ and $h_{t}^m$ denote the hidden states at time step $t$ of two RNNs learned by Eq. (6) and Eq. (7) respectively. In this work, we study four strategies to calculate the score function $g$, integrating MF and RNN. The details are described below.

PLASTIC-V1 This is a hard mechanism, using a simple way to calculate the mixing score from MF and RNN with the following formulation:

$$r_{ij,t} = g(e_{ij}^u, e_{ij}^m, h_{t}^u, h_{t}^m) = \frac{1}{1 + \exp(-s)} \quad (9)$$

$$s = e_{ij}^u \cdot h_{t}^u + e_{ij}^m \cdot h_{t}^m + b_i + b_j \quad (10)$$

where $b_i^u$ and $b_j^m$ are the biases of user $i$ and item $j$; $h_{t}^u$ and $h_{t}^m$ are computed by Eq. (6) and Eq. (7).

In fact, PLASTIC-V1 does not exploit the global factors in learning the temporal dynamics. In this paper, we also design a soft mixture mechanism and provide three strategies to account for the global factors $e_{ij}^u$ and $e_{ij}^m$ in learning $h_{t}^u$ and $h_{t}^m$, as described below (i.e., PLASTIC-V2, PLASTIC-V3 and PLASTIC-V4).

PLASTIC-V2 We use the latent factors of user $i$ ($e_{ij}^u$) and item $j$ ($e_{ij}^m$) pre-trained by MF model to initialize the hidden states of the LSTM cells $h_{0}^u$ and $h_{0}^m$ respectively.

PLASTIC-V3 We extend PLASTIC-V2 by treating $e_{ij}^u$ (for user $i$) and $e_{ij}^m$ (for item $j$) as the static context vectors, and feed them as an extra input into the computation of the temporal hidden states of users and items by LSTM. At each time step, the context information assists the inference of the hidden states of LSTM model.

PLASTIC-V4 This method uses an attention mechanism to compute a weight for each hidden state by exploiting the global factors. The mixing scores at time $t$ can be reformulated by:

$$r_{ij,t} = g(e_{ij}^u, e_{ij}^m, h_{t-1}^u, h_{t-1}^m, c_{i,t}^u, c_{j,t}^m) = \frac{1}{1 + \exp(-s)} \quad (11)$$

$$s = e_{ij}^u \cdot e_{ij}^m + h_{t}^u + h_{t}^m + b_i + b_j \quad (12)$$

where $c_{i,t}^u$ and $c_{j,t}^m$ are the context vectors at time step $t$ for user $i$ and item $j$; $h_{t}^u$ and $h_{t}^m$ are the hidden states of LSTMs at time step $t$, computed by:

$$h_{t}^u = \text{LSTM}(h_{t-1}^u, z_{it}^u, c_{i,t}^u) \quad (13)$$
$$h_{t}^m = \text{LSTM}(h_{t-1}^m, z_{jt}^m, c_{j,t}^m) \quad (14)$$

The context vectors $c_{i,t}^u$ and $c_{j,t}^m$ act as extra input in the computation of the hidden states in LSTMs to make sure that
every time step of the LSTMs can get full information of the context (long-term information). The context vectors $c_{i,t}^u$ and $c_{j,t}^m$ are the dynamic representations of the relevant long-term information for user $i$ and item $j$ at time $t$, calculated by

$$c_{i,t}^u = \sum_{k=1}^{U} a_{k,i}^t e_{k,i}^u; \quad c_{j,t}^m = \sum_{p=1}^{M} \beta_{p,j}^t e_{p,j}^m$$

(15)

where $U$ and $M$ are the numbers of users and items. The attention weights $a_{k,i}^t$ and $\beta_{p,j}^t$ for user $i$ and item $j$ at time step $t$ are computed by

$$a_{k,i}^t = \frac{\exp(\sigma(h_{i,t-1}^u, e_{k,i}^u))}{\sum_{k'=1}^U \exp(\sigma(h_{i,t-1}^u, e_{k',i}^u))}$$

(16)

$$\beta_{p,j}^t = \frac{\exp(\sigma(h_{j,t-1}^m, e_{p,j}^m))}{\sum_{p'=1}^M \exp(\sigma(h_{j,t-1}^m, e_{p',j}^m))}$$

(17)

where $\sigma$ is a feed-forward neural network to produce a real-valued score. The attention weights $a_{k,i}^t$ and $\beta_{p,j}^t$ together determine which user and item factors should be selected to generate $r_{i,j,t}$.

### 3.4 Generative Adversarial Network (GAN) for Recommendation

Generative adversarial network (GAN) [Goodfellow et al., 2014] consists of a generator $G$ and a discriminator $D$ that compete in a minimax game with two players: The discriminator tries to distinguish real high-rated items on training data from ranking or recommendation list predicted by $G$, and the generator tries to fool the discriminator to generate (predict) well-ranked recommendation list. Concretely, $D$ and $G$ play the following game on $V(D,G)$:

$$\min_G \max_D v(D,G) = \mathbb{E}_{x \sim P_{\text{true}}(x)}[\log D(x)] + \mathbb{E}_{z \sim P_{\text{noise}}(z)}[\log (1 - D(G(z)))]$$

(18)

Here, $x$ is the input data from training set, $z$ is the noise variable sampled from normal distribution.

We propose an adversarial framework to iteratively optimize two models: the generative model $G$ predicting recommendation list given historical user-item interactions, and the discriminative model $D$ predicting the relevance of the generated list. Like the standard generative adversarial networks (GANs) [Goodfellow et al., 2014], our model also optimizes the two models with a minimax two-player game. $D$ tries to distinguish the real high-rated items in the training data from the recommendation list generated by $G$, while $G$ maximizes the probability of $D$ making a mistake. Hopefully, this adversarial process can eventually adjust $G$ to generate plausible and high-quality recommendation list. We further elaborate the generator and discriminator below.

#### Discriminative Model

As depicted in Figure 1 (right side), we implement the discriminative $D$ via a Siamese Network that incorporates long and session-based ranking models in a pair-wise scenario. The discriminator $D$ has two symmetrical point-wise networks that share parameters and are updated by minimizing a pair-wise loss.

The objective of discriminator $D$ is to maximize the probability of correctly distinguishing the ground truth items from generated recommendation items. For $G$ fixed, we obtain the optimal parameters for the discriminator $D$ with the following formulation:

$$\theta^* = \arg\max_{\theta} \sum_{i \in \mathcal{U}} (E_{m_{-} \sim \text{true}, m_{+} \sim \text{true}} \log D_{\theta}(u_i, m_{-}, m_{+} | t) + E_{m_{+} \sim \text{true}, m_{-} \sim G} \log (1 - D_{\theta}(u_i, m_{-}, m_{+} | t)))$$

(19)

where $\mathcal{U}$ denotes the user set, $u_i$ denotes user $i$, $m_{+}$ is a positive (high-rating) item, $m_{-}$ is a negative item randomly chosen from the entire negative (low-rating) item space, $\theta$ and $\phi$ are parameters of $D$ and $G$, and $m_{g,t}$ is the generated item by $G$ given time $t$. Here, we adopt hinge loss as our training objective since it performs better than other training objectives. Hinge loss is widely adopted in various learning to rank scenario, which aims to penalize the examples that violate the margin constraint:

$$D(u_i, m_{-}, m_{+} | t) = \max \left\{ 0, \epsilon - g(e_{i}^{u}, e_{m_{+}}, h_{t}^{g}, h_{t}^{m_{+}}), \right\}$$

(20)

where $\epsilon$ is the hyper-parameter determining the margin of hinge loss, and we compress the outputs to the range of $(0, 1)$.

#### Generative Model

Similar to conditional GANs proposed in [Mirza and Osindero, 2014], our generator $G$ takes in the auxiliary information (user $u_i$ and time $t$) as input, and generates the ranking list for user $i$. Specifically, when $D$ is optimized and fixed after computing Eq. 19, the generator $G$ can be optimized by minimizing the following formulation:

$$\phi^* = \arg\min_{\phi} \sum_{m \in \mathcal{M}} (E_{m \sim \text{true}} \log (1 - D_{\theta}(u_i, m_{g,t}, m_{+} | t)))$$

(21)

Here, $\mathcal{M}$ denotes the item set. As in [Goodfellow et al., 2014], instead of minimizing $\log(1 - D(u_i, m_{g,t}, m_{+} | t))$, we train $G$ to maximize $\log(D(u_i, m_{g,t}, m_{+} | t))$.

#### Policy Gradient

Since the sampling of recommendation list by generator $G$ is discrete, it cannot be directly optimized by gradient descent as in the standard GAN formulation. Therefore, we use policy gradient based reinforcement learning algorithm [Sutton et al., 2000] to optimize the generator $G$ so as to generate highly rewarded recommendation list. Concretely, we have the following derivations:

$$\nabla_{\phi} J_G(u_i) = \nabla_{\phi} \mathbb{E}_{m_{g,t} \sim G} \log D_{\phi}(u_i, m_{g,t}, m_{+} | t)$$

(22)

$$= \sum_{m \in \mathcal{M}} \nabla_{\phi} G_{\phi}(m | u_i, t) \log D_{\phi}(u_i, m_{g,t}, m_{+} | t)$$

$$= \sum_{m \in \mathcal{M}} G_{\phi}(m | u_i, t) \nabla_{\phi} \log G_{\phi}(m | u_i, t) \log D_{\phi}(u_i, m_{g,t}, m_{+} | t)$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \nabla_{\phi} \log G_{\phi}(m_{k} | u_i, t) \log D_{\phi}(u_i, m_{k}, m_{+} | t)$$

where $K$ is number of items sampled by the current version of generator and $m_{k}$ is the $k$-th sampled item. With reinforcement learning terminology, we treat the term \( \log D(u_i, m_{k}, m_{+} | t) \) as the reward at time step $t$, and take an action $m_{k}$ at each time step. To accelerate the convergence,
**Algorithm 1:** Long and Session-based Ranking Model with Adversarial Network

1. **Input:** generator \( G_\phi \), discriminator \( D_\phi \), training data \( S \).
2. Initialize models \( G_\phi \) and \( D_\phi \) with random weights, and pre-train them on training data \( S \).
3. **repeat**
   4. **for** \( g \)-steps do
      5. Generate recommendation list for user \( i \) at time \( t \) using the generator \( G_\phi \).
      6. Sample \( K \) candidates from recommendation list.
      7. **for** \( k \in \{1, \ldots, K\} \) do
         8. Sample a positive item \( m_+ \) from \( S \).
         9. Compute the reward \( \log D(u_i, m_k, m_+ | t) \) with Eq.(20)
      10. Update generator \( G_\phi \) via policy gradient Eq.(22).
   11. **for** \( d \)-steps do
      12. Use current \( G_\phi \) to generate a negative item and combined with a positive item sampled from \( S \).
      13. Update discriminator \( D_\phi \) with Eq.(19).
   14. **until** convergence

... the rewards within a batch are normalized with a Gaussian distribution to make the significant differences.

The overall procedure is summarized in Algorithm 1. During the training stage, the discriminator and the generator are trained alternatively in an adversarial manner via Eq.(19) and Eq.(22), respectively.

## 4 Experimental Setup

### 4.1 Datasets

In order to evaluate the effectiveness of our model, we conduct experiments on two widely-used real-life datasets: Movielens-100K and Netflix (called "Netflix-Full"). For each dataset, we split the whole data into several training and testing intervals based on time, as done in [Wu et al., 2017], to simulate the actual situation of predicting future behaviors of users given the data that occurred strictly before current time step. Then, each testing interval is randomly divided into a validation set and a testing set. We removed the users and items that do not appear in training set from the validation and test sets. The detailed statistics are presented in Table 1.\(^2\)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Movielens-100K</th>
<th>Netflix-Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>943</td>
<td>480,189</td>
</tr>
<tr>
<td>Items</td>
<td>1,6831</td>
<td>17,770</td>
</tr>
<tr>
<td>Ratings</td>
<td>100,000</td>
<td>100,480,507</td>
</tr>
<tr>
<td>Train Data</td>
<td>09/97-03/98</td>
<td>12/99-11/05</td>
</tr>
<tr>
<td>Test Data</td>
<td>03/98-04/98</td>
<td>12/05</td>
</tr>
<tr>
<td>Train Ratings</td>
<td>77,714</td>
<td>98,074,901</td>
</tr>
<tr>
<td>Test Ratings</td>
<td>21,875</td>
<td>2,405,578</td>
</tr>
<tr>
<td>Sparsity [Wu et al., 2017]</td>
<td>0.063</td>
<td>0.012</td>
</tr>
</tbody>
</table>

### 4.2 Implementation Details

**Matrix Factorization.** We use matrix factorization with 5 and 16 factor numbers for Movielens and Netflix respectively [Wang et al., 2017]. The parameters are randomly initialized by a uniform distribution ranging in [-0.05, 0.05]. We take gradient-clipping to suppress the gradient to the range of [0,2,0.2]. L2 regularization (with \( \lambda = 0.05 \)) is used to the weights and biases of user and item factors.

**Recurrent Neural Network.** We use a single-layer LSTM with 10 hidden neurons, 15-dimensional input embeddings, and 4-dimensional dynamic states where each state contains 7-days users/items behavioral trajectories. That is, we take one month as the length of a session. The parameters are initialized in the same way as in MF. L2 regularization (with \( \lambda = 0.05 \)) is used to the weights and biases of the LSTM layer to avoid over-fitting.

**Generative Adversarial Nets.** We pre-train \( G \) and \( D \) on the training data with a pair-wise scenario, and use SGD algorithm with learning rate \( 1 \times 10^{-4} \) to optimize its parameters. In addition, we use matrix factorization model to generate 25 candidate items, and then re-rank these items with LSTM. In all experiments, we conduct mini-batch training with batch size 128.

### 4.3 Evaluation Metrics

To quantitatively evaluate our method, we adopt the rank-based evaluation metrics to measure the performance of top-\( n \) recommendation [Liu and Yang, 2008], including Precision@N, Normalised Discounted Cumulative Gain (NDCG@N), Mean Average Precision (MAP) and Mean Reciprocal Ranking (MRR).

### 4.4 Comparison to Baselines

In the experiments, we compare our approach with several strong baseline methods including Bayesian Personalised Ranking (BPR) [Rendle et al., 2009], Pairwise Ranking Factorization Machine (PRFM) [Qiang et al., 2013], LambdaFM [Yuan et al., 2016], Recurrent Recommender Networks (RRN) [Wu et al., 2017], and IRGAN [Wang et al., 2017]. For all the baseline methods, we adopt the default settings for all hyperparameters.

## 5 Experimental Results

### 5.1 Quantitative Results

We first evaluate the performance of top-\( n \) recommendation. The experimental results are summarized in Tables 1 and 3. Our model substantially and consistently outperforms the baseline methods by a noticeable margin on all the experimental datasets. In particular, we have explored several...
Table 2: Movie recommendation results (MovieLens).

<table>
<thead>
<tr>
<th>Metric</th>
<th>PR</th>
<th>PRFM</th>
<th>LambdaFM</th>
<th>RRN</th>
<th>IRGAN</th>
<th>PLASTIC-V1</th>
<th>PLASTIC-V2</th>
<th>PLASTIC-V3</th>
<th>PLASTIC-V4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@3</td>
<td>0.2910</td>
<td>0.2761</td>
<td>0.2550</td>
<td>0.2871</td>
<td>0.2814</td>
<td>0.2743</td>
<td>0.2491</td>
<td>0.2491</td>
<td>0.2491</td>
</tr>
<tr>
<td>P@5</td>
<td>0.3090</td>
<td>0.2894</td>
<td>0.2676</td>
<td>0.2975</td>
<td>0.3117</td>
<td>0.2883</td>
<td>0.2518</td>
<td>0.2518</td>
<td>0.2518</td>
</tr>
<tr>
<td>P@10</td>
<td>0.3497</td>
<td>0.3006</td>
<td>0.2912</td>
<td>0.3240</td>
<td>0.3312</td>
<td>0.3092</td>
<td>0.2864</td>
<td>0.2864</td>
<td>0.2864</td>
</tr>
<tr>
<td>MRR</td>
<td>0.3001</td>
<td>0.2817</td>
<td>0.2587</td>
<td>0.2624</td>
<td>0.2624</td>
<td>0.2573</td>
<td>0.2624</td>
<td>0.2624</td>
<td>0.2624</td>
</tr>
</tbody>
</table>

Table 3: Movie recommendation results (Netflix-Full).

<table>
<thead>
<tr>
<th>Metric</th>
<th>PR</th>
<th>PRFM</th>
<th>LambdaFM</th>
<th>RRN</th>
<th>IRGAN</th>
<th>PLASTIC-V1</th>
<th>PLASTIC-V2</th>
<th>PLASTIC-V3</th>
<th>PLASTIC-V4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@3</td>
<td>0.3098</td>
<td>0.2870</td>
<td>0.2693</td>
<td>0.3004</td>
<td>0.3049</td>
<td>0.3092</td>
<td>0.2848</td>
<td>0.2848</td>
<td>0.2848</td>
</tr>
<tr>
<td>P@5</td>
<td>0.3367</td>
<td>0.3137</td>
<td>0.2789</td>
<td>0.3304</td>
<td>0.3351</td>
<td>0.3367</td>
<td>0.3192</td>
<td>0.3192</td>
<td>0.3192</td>
</tr>
<tr>
<td>P@10</td>
<td>0.3786</td>
<td>0.3502</td>
<td>0.3280</td>
<td>0.3786</td>
<td>0.3831</td>
<td>0.3502</td>
<td>0.3280</td>
<td>0.3280</td>
<td>0.3280</td>
</tr>
<tr>
<td>MRR</td>
<td>0.3319</td>
<td>0.3026</td>
<td>0.2810</td>
<td>0.3304</td>
<td>0.3351</td>
<td>0.3319</td>
<td>0.3131</td>
<td>0.3131</td>
<td>0.3131</td>
</tr>
</tbody>
</table>

Table 4: The recalled items from top-10 candidates.

The main strength of our model comes from its capability of prioritizing both long-term and short-term information in top-n recommendation. In addition, our mixture mechanisms (hard and soft) also seem quite effective to integrate MF and RNN.

To better understand the adversarial training process, we visualize the learning curves of PLASTIC-V4 as shown in Figure 2. Due to the limited space, we only report the Precision@5 and NDCG@5 scores as in [Wang et al., 2017]. The other metrics exhibit a similar trend. As shown in Figure 2, after about 50 epochs, both Precision@5 and NDCG@5 converge and the winning player is the generator which is used to generate recommendation list for our final top-n item recommendation. The performance of generator G becomes better with the effective feedback (reward) from discriminator D. On the other hand, once we have a set of high-quality recommendation movies, the performance of D deteriorates gradually in the training procedure and makes mistakes for predictions. In our experiments, we use the generator G with the best performance to predict test data.

5.2 Case study

In this section, we will further show the advantages of our models through some quintessential examples. In Table 4, we provide the recommendation lists generated by two strong competitors (i.e., IRGAN and RNN) as well as the proposed PLASTIC-V4 model for two users who are randomly selected from the Netflix-Full dataset. Our model can rank the positive movies in higher positions than compared methods. For example, some emerging movies such as “Session 9” and “The Last Valley” that are truly attractive to the first user have been recommended by our models, whereas they are ignored by the compared methods. In fact, we can include all positive recommended by our models, whereas they are ignored by the compared methods.

6 Conclusion

In this paper, we proposed a novel adversarial process for top-n recommendation. Our models incorporate both matrix factorization and recurrent neural network to exploit the benefits of the long-term and short-term knowledge. Experiments on two real-life datasets showed the performance superiority of our models for top-n recommendation.

Figure 2: Learning curves of GAN on Netflix-Full.
References


