DeepSet: Deep Recommendation with Setwise Preference

Lin Li\textsuperscript{1,2}, Wei Ke Pan\textsuperscript{1,3}\textsuperscript{*}, Guanliang Chen\textsuperscript{2} and Zhong Ming\textsuperscript{1,3}\textsuperscript{*}

\textsuperscript{1} College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China
\textsuperscript{2} Centre for Learning Analytics at Monash (CoLAM), Monash University, Melbourne, Australia,
\textsuperscript{3} Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ), Shenzhen, China.
\{lin.li,guanliang.chen\}@monash.edu, \{panweike,mingz\}@szu.edu.cn
Motivation

Most of existing neural recommendation methods [He et al., 2017, Yang et al., 2021] are essentially defining new prediction rules or just neutralizing the typical user and item factors by stacking non-linear neural computations, which may not guarantee better performance and increase the model complexity instead.

One obvious limitation is that they adopt classical preference assumptions on single items or item pairs.
Overall of Our Solution

- We employ setwise preference motivated by CoFiSet [Pan et al., 2019] into neural model learning process, which can further facilitate the real potential of neural networks on handling complex patterns.

- Specifically, we devise a novel general solution for deep recommendation with MLP to model setwise preference, i.e., DeepSet, and derive three variants to incorporate this novel assumption by activating the setwise preference at different neural layers, namely the feature input layer, the feature output layer, and the prediction layer, corresponding to DeepSet(fi), DeepSet(fo) and DeepSet(p), respectively.
Advantages of Our Solution

- DeepSet is **vertical to existing deep recommendation models** based on or integrated with MLP, which means that DeepSet can be instantiated with various MLP-based recommendation methods.

- Extensive experiments conducted on **four public datasets** demonstrate the effectiveness of our solution over the neural collaborative filtering framework.
**Related Work**

**Neural Collaborative Filtering [He et al., 2017]** is a representative neural one-class collaborative filtering model containing two components, i.e., generalized matrix factorization (GMF) and multi-layer perceptron (MLP), which can be also used to make prediction seperately.

- **GMF**
  \[
  \hat{r}_{ui}^G = \sigma((U_{u:}^G \odot V_{i:}^G)w_T^G).
  \]  
  \[ (1) \]

- **MLP**
  \[
  \hat{r}_{ui}^M = \sigma(f_{mlp}(U_{u:}^M, V_{i:}^M)w_T^M),
  \]  
  \[ (2) \]

The **prediction rule** of NeuMF is as follows:

\[
\hat{r}_{ui} = \sigma([U_{u:}^G \odot V_{i:}^G, f_{mlp}(U_{u:}^M, V_{i:}^M)])w^T).
\]  
\[ (3) \]
NeuMF adopts the preference assumption defined on single items under the pointwise learning scheme

\[ r_{ui} = 1, r_{uj} = 0, i \in \mathcal{I}_u, j \in \mathcal{I} \setminus \mathcal{I}_u, \quad (4) \]

The corresponding loss function of NeuMF is as follows:

\[
\mathcal{L} = - \sum_{(u,i) \in \mathcal{R}^+ \cup \mathcal{R}^-} r_{ui} \ln(\hat{r}_{ui}) + (1 - r_{ui}) \ln(1 - \hat{r}_{ui}).
\]

\[
(5)
\]
Preference Assumption: Motivated by the promising improvements achieved by the setwise preference assumption proposed in [Pan et al., 2019], we refine the preference assumption on single items underlying the learning process of NeuMF:

$$r_{uP} = 1, r_{uj} = 0, P \subseteq I_u, j \in A, A \subseteq I \setminus I_u,$$

where $P$ is a set of observed items randomly sampled from user $u$’s history records, $A$ is a set of unobserved items randomly sampled from all the items excluding $u$’s interacted ones.
We attempt to incorporate the effect of this new assumption by activating the setwise preference at different layers of NeuMF, and further derive three variants, namely

- **DeepSet(fi)**: setwise preference activated at the feature input layer.
- **DeepSet(fo)**: setwise preference activated at the feature output layer.
- **DeepSet(p)**: setwise preference activated at the prediction layer.
Figure 1: DeepSet.
In this variant, the setwise preference is activated at the feature input layer, which means that we aggregate the item embeddings of the corresponding item-set, and then feed the aggregated item-set embedding into the following layers.

- **DeepSet(fi, GMF)**

  \[
  \hat{r}_{uP}^G = \sigma((U_u^G \odot \bar{V}_P^G)w_G^T),
  \]
  \[
  (7)
  \]

- **DeepSet(fi, NeuMF)**

  \[
  \hat{r}_{uP} = \sigma([U_u^G \odot \bar{V}_P^G, f_{mlp}([U_u^M, \bar{V}_P^M])]w^T),
  \]
  \[
  (8)
  \]
DeepSet(p) merges the final predicted score of each item in a sampled set at the last layer of the network, i.e., the prediction layer. Specifically, each (user, item) pair in the set actually shares the whole network and would be computed in parallel, which means that each item in the set will get a prediction score, which would be averaged to get the set score.

- DeepSet(p, GMF)
  \[
  \hat{r}_{uP}^G = \frac{1}{|P|} \sum_{i \in P} \sigma\left((U_u^G \odot V_i^G)w_G^T\right).
  \]  
  \[ (9) \]

- DeepSet(p, NeuMF)
  \[
  \hat{r}_{uP} = \frac{1}{|P|} \sum_{i \in P} \sigma([U_u^G \odot V_i^G, f_{\text{mlp}}([U_u^M, V_i^M])])w^T.
  \]  
  \[ (10) \]
DeepSet(fo) serves as a compromise between DeepSet(fi) and DeepSet(p), which activates the setwise average pooling operation at the last feature computation layer right before the prediction layer, i.e., the feature output layer.

- **DeepSet(fo, NeuMF):**

\[
\hat{r}_u^p = \sigma\left(\left[ U_u^G \odot \bar{V}^G_p, \frac{1}{|P|} \sum_{i \in P} f^{mlp}\left(\left[ U_i^M, V_i^M \right]\right)\right] w^T\right). \tag{11}
\]
With our proposed preference assumption, the learning samples defined on the item-level have been changed to those on the set-level. Therefore, the main difference of loss function between DeepSet and NeuMF is that the loss is extended to the setwise preference from that of single items.

\[
\min_{\Theta} - \sum_{u \in U} \left( \sum_{\mathcal{P} \subseteq I_u} \ln(\hat{r}_{uP}) + \sum_{\mathcal{A} \subseteq \mathcal{I} \setminus I_u} \sum_{j \in \mathcal{A}} \ln(1 - \hat{r}_{uj}) \right). \tag{12}
\]
Time Complexity

We analyze the complexity of DeepSet using the tensor shapes, from which we can see the time cost for the DeepSet-NeuMF models would be: $T_{\text{DeepSet}(\text{fi, NeuMF})} < T_{\text{DeepSet}(\text{fo, NeuMF})} \approx T_{\text{DeepSet}(\text{p, NeuMF})} \leq |\mathcal{P}| T_{\text{NeuMF}}$.

Table 1: The tensor shapes participated in each computation for training DeepSet(fi, GMF), DeepSet(p, GMF), DeepSet(fi, NeuMF), DeepSet(fo, NeuMF) and DeepSet(p, NeuMF) with set size $|\mathcal{P}|$ on one batch data. When $|\mathcal{P}| = 1$, DeepSet(fi, GMF) and DeepSet(p, GMF) both reduce to GMF, and DeepSet(fi, NeuMF), DeepSet(fo, NeuMF) and DeepSet(p, NeuMF) reduce to NeuMF.

<table>
<thead>
<tr>
<th>Methods</th>
<th>elementwise-product</th>
<th>matrix-multiplication</th>
<th>reduce_mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeepSet(fi, GMF)</td>
<td>$\mathbb{R}^{bs \times</td>
<td>\mathcal{P}</td>
<td>\times d_G}$</td>
</tr>
<tr>
<td>DeepSet(p, GMF)</td>
<td>$\mathbb{R}^{bs \times</td>
<td>\mathcal{P}</td>
<td>\times d_G}$</td>
</tr>
<tr>
<td>DeepSet(fi, NeuMF)</td>
<td>$\mathbb{R}^{bs \times</td>
<td>\mathcal{P}</td>
<td>\times d_G}$, $\mathbb{R}^{bs \times</td>
</tr>
<tr>
<td>DeepSet(fo, NeuMF)</td>
<td>$\mathbb{R}^{bs \times</td>
<td>\mathcal{P}</td>
<td>\times d_G}$, $\mathbb{R}^{bs \times</td>
</tr>
<tr>
<td>DeepSet(p, NeuMF)</td>
<td>$\mathbb{R}^{bs \times</td>
<td>\mathcal{P}</td>
<td>\times d_G}$, $\mathbb{R}^{bs \times</td>
</tr>
</tbody>
</table>

Notice that $d_G$ is the input dimension of user and item for the GMF part, $d_M$ is for the MLP part.
Datasets

We conduct experiments on four public datasets with different sparsity, including three movie datasets MovieLens100K, MovieLens1M, Netflix5K5K and one tag dataset UserTag, which are the same as that of [Pan et al., 2019].

Table 2: Statistic of the datasets used in the experiments ($\mathcal{R}^{tr}$, $\mathcal{R}^{va}$ and $\mathcal{R}^{te}$ denote the training data, validation data and test data, respectively).

| Dataset              | #user | #items | $|\mathcal{R}^{tr} \cup \mathcal{R}^{va}|$ | $|\mathcal{R}^{te}|$ | density     |
|----------------------|-------|--------|----------------------------------------|----------------------|-------------|
| MovieLens100K        | 943   | 1,682  | 27,688                                 | 27,687               | 1.7456%     |
| MovieLens1M          | 6,040 | 3,953  | 287,641                                | 287,640              | 1.2050%     |
| UserTag              | 3,000 | 2,000  | 123,128                                | 123,218              | 2.0521%     |
| Netflix5K5K          | 5,000 | 5,000  | 77,936                                 | 77,936               | 0.3117%     |

- We first randomly sample out one half of the observed (user, item) pairs in the datasets, and treat the remaining half as the test data.
- Then we randomly take one (user, item) pair for each user on average from the sampled half data as the validation data and the remaining part of this half as the training data.
Evaluation Metrics

- We use five commonly used ranking-oriented evaluation metrics, including precision, recall, F1, NDCG and 1-call.
- We rank all the unobserved items for each user for fair comparisons among all the methods.
- We truncate a ranking list at 5 for all the metrics, i.e., precision@5, recall@5, F1@5, NDCG@5 and 1-call@5 as main results.
Baseline

- **LogisticMF** [Johnson, 2014] is a MF-based method with pointwise learning scheme.
- **BPR** [Rendle et al., 2009] builds upon MF by maximizing the pairwise preference difference between an observed pair and an unobserved one.
- **GMF** [He et al., 2017] is a shallow network extended on LogisticMF with preference predicted via a “weighted” inner product.
- **NeuMF** [He et al., 2017] is a deep learning based OCCF method with logistic loss, integrating GMF with an MLP to make prediction.
- **SQL-RANK** [Wu et al., 2018] is a recent non-neural OCCF method that adopts a listwise preference assumption.
- **RBM-OCCF** [Jahrer and Töscher, 2012, Salakhutdinov et al., 2007] uses restricted Boltzmann machines to model the one-class data.
We implement Logistic MF, BPR and our DeepSet with TensorFlow\(^1\) and use Adam [Kingma and Ba, 2015] as the optimizer.

For fair comparison, we keep the dimension of the user’s latent vector and item’s latent vector used to yield inner product or element-wise product as 20.

For Logistic MF, BPR and our DeepSet, we select the learning rate from \{0.0001, 0.0005, 0.001, 0.005\} and the batch size from \{128, 256, 512, 1024\} according to the NDCG@5 performance on the validation data.

For SQL-RANK and RBM-OCCF, we select the related parameters under the guidance of original papers, which includes learning rate, decay rate and regularization parameter.

\(^1\)https://www.tensorflow.org/
For NeuMF and our DeepSet extension upon NeuMF, the embedded multi-layer network are all fixed as 4-layer with 64 → 32 → 16 → 8 structure, i.e., the dimension of the user’s and item’s embedding fed to MLP is 32.

The negative sampling ratio for all the methods except BPR is fixed as 3.

We fix the set size at 3 for all the variants of our DeepSet.
Table 3: Recommendation performance of six baseline methods and five derived methods of our DeepSet upon GMF and NeuMF.

<table>
<thead>
<tr>
<th>Method</th>
<th>MovieLens100K</th>
<th></th>
<th></th>
<th></th>
<th>MovieLens1M</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision@5</td>
<td>Recall@5</td>
<td>F1@5</td>
<td>NDCG@5</td>
<td>1-call@5</td>
<td>Precision@5</td>
<td>Recall@5</td>
<td>F1@5</td>
</tr>
<tr>
<td>LogisticMF</td>
<td>0.3769 ±0.0096</td>
<td>0.0928</td>
<td>0.1298</td>
<td>0.3936</td>
<td>0.8075 ±0.0090</td>
<td>0.4151 ±0.0053</td>
<td>0.0667</td>
<td>0.1029</td>
</tr>
<tr>
<td>BPR</td>
<td>0.3835 ±0.0017</td>
<td>0.0949</td>
<td>0.1329</td>
<td>0.3995</td>
<td>0.8164 ±0.0107</td>
<td>0.4163 ±0.0036</td>
<td>0.0654</td>
<td>0.1017</td>
</tr>
<tr>
<td>SQL-RANK</td>
<td>0.3977 ±0.0131</td>
<td>0.1025</td>
<td>0.1415</td>
<td>0.4215</td>
<td>0.8315 ±0.0200</td>
<td>0.4343 ±0.0010</td>
<td>0.0694</td>
<td>0.1075</td>
</tr>
<tr>
<td>RBM-OCCF</td>
<td>0.3792 ±0.0080</td>
<td>0.0927</td>
<td>0.1306</td>
<td>0.3966</td>
<td>0.8242 ±0.0133</td>
<td>0.3873 ±0.0073</td>
<td>0.0590</td>
<td>0.0922</td>
</tr>
<tr>
<td>GMF</td>
<td>0.3651 ±0.0059</td>
<td>0.1027</td>
<td>0.1280</td>
<td>0.3799</td>
<td>0.8028 ±0.0112</td>
<td>0.3869 ±0.0027</td>
<td>0.0609</td>
<td>0.0943</td>
</tr>
<tr>
<td>NeuMF</td>
<td>0.3853 ±0.0015</td>
<td>0.0961</td>
<td>0.1336</td>
<td>0.4058</td>
<td>0.8139 ±0.0130</td>
<td>0.4470 ±0.0043</td>
<td>0.0719</td>
<td>0.1111</td>
</tr>
<tr>
<td>NeuMF</td>
<td>0.3825 ±0.0160</td>
<td>0.0970</td>
<td>0.1343</td>
<td>0.4016</td>
<td>0.8246 ±0.0094</td>
<td>0.4012 ±0.0054</td>
<td>0.0658</td>
<td>0.1010</td>
</tr>
<tr>
<td>DeepSet(fl, GMF)</td>
<td>0.3616 ±0.0076</td>
<td>0.0927</td>
<td>0.1279</td>
<td>0.3799</td>
<td>0.8128 ±0.0101</td>
<td>0.4065 ±0.0064</td>
<td>0.0646</td>
<td>0.1000</td>
</tr>
<tr>
<td>DeepSet(p, GMF)</td>
<td>0.3659 ±0.0086</td>
<td>0.0924</td>
<td>0.1278</td>
<td>0.3848</td>
<td>0.8142 ±0.0164</td>
<td>0.4033 ±0.0010</td>
<td>0.0655</td>
<td>0.1008</td>
</tr>
<tr>
<td>DeepSet(fl, NeuMF)</td>
<td>0.3731 ±0.0153</td>
<td>0.0930</td>
<td>0.1295</td>
<td>0.3906</td>
<td>0.8203 ±0.0202</td>
<td>0.4199 ±0.0154</td>
<td>0.0674</td>
<td>0.1040</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>UserTag</th>
<th></th>
<th></th>
<th></th>
<th>Netflix5K5K</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision@5</td>
<td>Recall@5</td>
<td>F1@5</td>
<td>NDCG@5</td>
<td>1-call@5</td>
<td>Precision@5</td>
<td>Recall@5</td>
<td>F1@5</td>
</tr>
<tr>
<td>LogisticMF</td>
<td>0.2862 ±0.0010</td>
<td>0.0430</td>
<td>0.0687</td>
<td>0.2951</td>
<td>0.5713 ±0.0553</td>
<td>0.2130 ±0.0020</td>
<td>0.0739</td>
<td>0.0881</td>
</tr>
<tr>
<td>BPR</td>
<td>0.2964 ±0.0024</td>
<td>0.0444</td>
<td>0.0708</td>
<td>0.3041</td>
<td>0.5951 ±0.0023</td>
<td>0.2305 ±0.0060</td>
<td>0.0780</td>
<td>0.0995</td>
</tr>
<tr>
<td>SQL-RANK</td>
<td>0.2806 ±0.0073</td>
<td>0.0422</td>
<td>0.0670</td>
<td>0.2891</td>
<td>0.5777 ±0.1398</td>
<td>0.1959 ±0.0271</td>
<td>0.0684</td>
<td>0.0816</td>
</tr>
<tr>
<td>RBM-OCCF</td>
<td>0.2753 ±0.0019</td>
<td>0.0408</td>
<td>0.0649</td>
<td>0.2875</td>
<td>0.5817 ±0.0050</td>
<td>0.2345 ±0.0090</td>
<td>0.0858</td>
<td>0.1006</td>
</tr>
<tr>
<td>GMF</td>
<td>0.2738 ±0.0036</td>
<td>0.0418</td>
<td>0.0682</td>
<td>0.2812</td>
<td>0.5513 ±0.0063</td>
<td>0.1885 ±0.0073</td>
<td>0.0637</td>
<td>0.0766</td>
</tr>
<tr>
<td>DeepSet(fl, GMF)</td>
<td>0.2851 ±0.0034</td>
<td>0.0434</td>
<td>0.0690</td>
<td>0.2943</td>
<td>0.5740 ±0.0041</td>
<td>0.2392 ±0.0052</td>
<td>0.0838</td>
<td>0.1003</td>
</tr>
<tr>
<td>DeepSet(p, GMF)</td>
<td>0.2834 ±0.0039</td>
<td>0.0435</td>
<td>0.0690</td>
<td>0.2925</td>
<td>0.5789 ±0.0068</td>
<td>0.2302 ±0.0089</td>
<td>0.0839</td>
<td>0.0980</td>
</tr>
<tr>
<td>NeuMF</td>
<td>0.2764 ±0.0024</td>
<td>0.0413</td>
<td>0.0662</td>
<td>0.2850</td>
<td>0.5699 ±0042</td>
<td>0.2035 ±0.0118</td>
<td>0.0669</td>
<td>0.0812</td>
</tr>
<tr>
<td>DeepSet(fl, NeuMF)</td>
<td>0.2652 ±0.0023</td>
<td>0.0401</td>
<td>0.0640</td>
<td>0.2731</td>
<td>0.5415 ±0.0088</td>
<td>0.1918 ±0.0016</td>
<td>0.0735</td>
<td>0.0845</td>
</tr>
<tr>
<td>DeepSet(p, NeuMF)</td>
<td>0.2790 ±0.0037</td>
<td>0.0432</td>
<td>0.0685</td>
<td>0.2897</td>
<td>0.5825 ±0.0061</td>
<td>0.2326 ±0.0023</td>
<td>0.0860</td>
<td>0.0997</td>
</tr>
<tr>
<td>DeepSet(fl, NeuMF)</td>
<td>0.2854 ±0.0023</td>
<td>0.0436</td>
<td>0.0693</td>
<td>0.2961</td>
<td>0.5806 ±0.0114</td>
<td>0.2363 ±0.0039</td>
<td>0.0799</td>
<td>0.0976</td>
</tr>
</tbody>
</table>
Among the four very competitive baselines, i.e., LogisticMF, BPR, SQL-RANK and RBM-OCCF, SQL-RANK performs best on MovieLens100K and MovieLens1M, while on UserTag BPR takes a lead and RBM-OCCF shows higher accuracy on Netflix5K5K. Notice that UserTag is a relatively dense dataset and Netflix5K5K instead is very sparse, which shows that listwise preference assumption may not be friendly to very dense or sparse data, and BPR can achieve fair performance in most cases, while RBM-OCCF may be better for sparse data.

GMF and NeuMF are actually outperformed by LogisticMF, BPR and SQL-RANK on all the four datasets, which shows that using neural networks to devise more elaborated prediction rules does not necessarily guarantee improved performance.
For DeepSet derived upon GMF, i.e., DeepSet(fi, GMF) and DeepSet(p, GMF), significantly outperform GMF on all the datasets, showing the effectiveness of our solution over shallow network like GMF. And we can see that activating the setwise preference at the feature input layer is more beneficial in this case.

DeepSet(fi, NeuMF) performs better than NeuMF on Movielens100K and MovieLens1M, but is slightly inferior on Netflix5K5K and UserTag. Notice that DeepSet(fi, NeuMF) activates the setwise preference at the feature input layer, which may introduce more noise when the data are too sparse or too dense. This may explain the inconsistency of DeepSet(fi,NeuMF) on UserTag and Netflix5K5K, since UserTag and Netflix5K5K are the most dense and sparse datasets, respectively.
Both DeepSet(fo, NeuMF) and DeepSet(p, NeuMF) perform substantially better than NeuMF on all the datasets, which shows that activating the setwise preference at the later layers of a deep network is more favorable to enhance the recommendation performance.

In particular, DeepSet(fo, NeuMF) achieves the highest improvements over NeuMF, which shows that activating the setwise preference at a middle layer may be most effective in improving the performance of the deep learning based recommendation, especially for models like NeuMF that comprises two comparably independent components.
Across the four datasets, the derived methods with our DeepSet indeed achieve better performance compared to those without setwise preference. Specifically, DeepSet performs better than BPR on MovieLens100K, MovieLens1M and Netflix5K5K, and slightly weaker on UserTag. From this perspective, our DeepSet achieves clearly better overall performance and is highly adaptive to different datasets.
Figure 2: Top-k (NDCG) recommendation performance of DeepSet-GMF group, i.e., GMF, DeepSet(fi, GMF) and DeepSet(p, GMF) with $|\mathcal{P}| = 3$. 
**Figure 3:** Top-$k$ (NDCG) recommendation performance of DeepSet-NeuMF group, i.e., NeuMF, DeepSet(fi, NeuMF), DeepSet(fo, NeuMF) and DeepSet(p, NeuMF) with $|\mathcal{P}| = 3$. 
For the performance of NDCG@\(k\) at different values of \(k\), the difference between different methods keep the same as that of \(k = 5\), showing the relative recommendation performance are similar with different values of \(k\).

For the DeepSet-GMF group and the DeepSet-NeuMF group, the new models derived with DeepSet achieve the best performance of NDCG@\(k\) at different values of \(k\) (i.e., \(1 \leq k \leq 10\)) on across all the four datasets, which again shows that our DeepSet indeed encourages the deep model (e.g., NeuMF) and the shallow model (e.g., GMF) to reach the full potential.
Figure 4: Recommendation performance of NeuMF and DeepSet(fo,NeuMF) with 4-layer MLP and 3-layer MLP, i.e., NeuMF-4, NeuMF-3, DeepSet-4 and DeepSet-3.
The 3-layer NeuMF performs slightly better than that with 4 layers on MovieLens100K, MovieLens1M and UserTag, while on Netflix5K5K using 4 layers are better than using 3 layers. Since the sparsity of Netflix5K5K is relatively high, we think this may show that a deeper network is more effective when the training data is highly sparse.

The performance of 3-layer DeepSet is close to DeepSet with MLP of 4 layers on all the datasets. This shows that DeepSet is not very sensitive to the changes of the number of neural layers.

DeepSet with fewer layers still significantly outperforms NeuMF with 4 layers or 3 layers, which shows the effectiveness of DeepSet upon deep learning based recommendation models like NeuMF.
Evaluation of Efficiency (1/2)

Figure 5: The CPU time of DeepSet(fi, GMF), DeepSet(p, GMF), DeepSet(fi, NeuMF), DeepSet(fo, NeuMF) and DeepSet(p, NeuMF) run on a Windows 10 machine with Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz 3.19GHz/32GB RAM. (batch size = 128, learning rate=0.0001, T=100)
The time cost of DeepSet(fi, GMF) and DeepSet(p, GMF) are very close at each value of $|\mathcal{P}|$, where DeepSet(fi, GMF) takes slightly more time, showing that for a shallow model like GMF, activating the setwise preference at the feature input layer cost a bit more time.

For the DeepSet-NeuMF group, the CPU time of the three methods are linearly related with set size $|\mathcal{P}|$. The time of DeepSet(fo, NeuMF) and DeepSet(p, NeuMF) are very close and significantly larger than that of DeepSet(fi, NeuMF), but the time cost by all of them increase less than the $|\mathcal{P}|$ multiples, which is consistent with our analysis of the time complexity. In summary, our DeepSet will not cost more time since $|\mathcal{P}|$ is usually a small integer such as 3.
Conclusions

- We propose a general solution DeepSet that refines the typical learning pattern on a single item with the preference assumption defined on item-sets.
- Under this new assumption, we propose to incorporate the effects of the setwise preference at the feature input layer, the feature output layer and the prediction layer of a neural model, leading to three variants, i.e., DeepSet(fi), DeepSet(fo) and DeepSet(p).
- Our solution effectively enhances neural recommendation methods e.g., NeuMF. On top of the improved performance, our solution would not introduce any new model parameters.
In the future, we plan to study the effectiveness of our solution on more neural recommendation methods and more types of user behaviors [Xia et al., 2021]. Moreover, it is also interesting to explore how to activate the setwise preference in a different way.
We thank the editors and reviewers for their encouraging words and constructive advices, as well as very detailed and expert comments.

We thank the support of National Natural Science Foundation of China Nos. 61872249 and 61836005.

Feel free to contact us if you have any questions.
Neural collaborative filtering.
WWW’17, pages 173–182.

Collaborative filtering ensemble for ranking.

Logistic matrix factorization for implicit feedback data.
In NeurIPS 2014 Workshop.

Adam: A method for stochastic optimization.
ICLR’15.

Personalized recommendation with implicit feedback via learning pairwise preferences over item-sets.

BPR: Bayesian personalized ranking from implicit feedback.
UAI’09, pages 452–461.

Restricted Boltzmann machines for collaborative filtering.
ICML’07, pages 791–798.

SQL-Rank: A listwise approach to collaborative ranking.
ICML’18, pages 5311–5320.

Knowledge-enhanced hierarchical graph Transformer network for multi-behavior recommendation. 
AAAI '21, pages 4486–4493.

Enhanced graph learning for collaborative filtering via mutual information maximization.