Generative Adversarial User Model for Reinforcement Learning Based Recommendation System PyTorch Implementation

Can Gözpınar [*1] Batu Helvacıoğlu [*2]

Abstract

There are many approaches to create a recommendation system. One of the most common solutions is to use reinforcement learning. To utilize reinforcement learning one needs an environment for the agent to interact with. In the setting of recommendation systems, this environment should mimic the user behaviour. (1) suggests using Generative Adversarial Networks to create this environment. Specifically they suggest using the discriminator as a reward function and the generator which as a greedy action selector to simulate the user behaviour. We modified this approach by adding a separate generator network which was not included in the original paper. We believe with more densely populated datasets our approach can be viable.

1. Introduction

Recommendation systems are beneficial to both the online content providers and their online users. It aims to expose the users to the content which they are more likely to engage with. Therefore, recommendation systems have become an inseparable part of almost every online content provider. There are many approaches to recommendation system architectures but most of them fail to establish the long-term relations between the recommendation system and the user such as recommendation systems ability to alter the users preferences in the long term. Therefore, to improve upon these assumptions one can use Reinforcement Learning since, it is capable of taking the users long-term interest into account. This stems from the fact that reinforcement learning agents are optimized to maximize the cumulative reward. Even though there are many promising reinforcement learning approaches which can be beneficial in this context, the unknown model of the user environment, and the in-feasibility of training a reinforcement learning agent online remains as obstacles. To solve this issue we propose the use of Generative Adversarial Networks (GANs) to learn the behavior of the user ”environment” which will then be used to train the reinforcement learning agent. This learned user model will be able to simulate user actions given a set of possible actions (display set). The accompanying reward function will return the rewards to the reinforcement learning agent.With this goal in mind, we are going to modify and implement the ”Generative Adversarial User Model for Reinforcement Learning Based Recommendation System” (1) paper using PyTorch.

[*]Equal contribution [1]Department of Computer Engineering [2]Department of Computer Engineering. Correspondence to: Can Gözpınar <cgozpinar18@ku.edu.tr>, Batu Helvacıoğlu <bhelvacioglu18@ku.edu.tr>.

COMP547 Deep Unsupervised Learning, Spring 2022.

2. Related Work

User recommendation algorithms usually use simple user models. Some models assume that user choices of each item are independent and use logistic regression to model the user. Wide&Deep networks (3), XGBOOST(2) and DFM (4) are examples to these models. Collaborative competitive filtering (8) does not consider each item independent but assumes each page view is independent from previous page views of the user. Session-based models like Session-based RNN (5) and session-based KNN (6) utilizes user history to avoid the previous assumption but cannot be used with reinforcement learning. LinUCB (7) is a Bandit based approaches that better handles adversaries but again cannot be used with reinforcement learning. Generative Adversarial User Model for Reinforcement Learning Based Recommendation System (1) suggests using GANs to model the user behaviour and also to obtain a reward function that can be used to simulate an environment for reinforcement learning agent training. Specifically the generator of the GAN is used for the user model while the discriminator is used for the reward function.

3. The Approach

3.1. Reinforcement Learning Formulation

We followed the approach proposed by the ”Generative Adversarial User Model for Reinforcement Learning Based Recommendation System” (1) with a few modifications. The paper proposes that Generative Adversarial Network

(GAN) architectures can be used to learn Deep Neural Network models which can be used as the Reward function as part of the environment in the Reinforcement learning (RL) formulation to train RL agents which can do recommendation system tasks. The learned Reward function would correspond to the Discriminator in the GAN formulation. In the Reinforcement Learning setting, the state will correspond to the previous items clicked by a certain user. Possible actions will be a combination of k possible items that will form a display set. The reinforcement learning agent will interact with the environment via the display sets that it creates. Reward will be the total utility of the display set for the user which will be given by the Discriminator (Reward Model) in the GAN formulation. Environment will consist of the reward function and the user model obtained from the GAN. Specified user actions are clicking a specific item in the display set or not clicking anything. As like the original paper, our focus was learning this Discriminator Reward Model via the GAN training. As for the GAN formulation, we again implemented the formulation proposed by (1).

3.2. Generative Adversarial Training

Our focus went into this part of formulation. The overall architecture consists of three neural network models. These are the history_lstm, Discriminator_Reward_Model, and Generator_User_Model.

3.2.1. History LSTM

The History LSTM is a BI-LSTM network which learns to encode the history of the user's interactions with the recomendation system to a meaningful latent vector. Here the inputs to the History LSTM are the vector representations of the user actions in a sequential order. For example, given a new item that the user has clicked as input, the history LSTM returns a new updated state representation. This user history representation should ideally be a vector which encompasses the user's personality. For example, in our implementation, the datasets we used did not have any meaningful information regarding the items characteristics and only had an unique id. Therefore, we had to restrict our item vectors to one hot encoding vectors. As a result, the history lstm took these one hot encoding representations of the items that the user has clicked at a given time as its input, and produced a vector which we used to encode that corresponding user's history. These state representations outputted by the history lstm will be fed into both the Discriminator Reward Model and the Generator User Model along with other things to output meaningful user tailored predictions. For example, Generator User Model would take this state representation to take the corresponding user's personality while trying to emulate that user's actions. Similarly, the Discriminator Reward Model would take the state representation to have a sense of what that corresponding user's personality is like in order to decide whether a given action is likely to be taken by a real user whose past interactions are encoded by that state vector representation. The History LSTM model is a BI-LSTM architecture. BI-LSTM's were chosen due to their ability of encoding past and processing sequential inputs which was crucial for us to encode past user interactions and update them at consecutive time steps with newly taken actions. Also, they usually yield better performance than RNNs which are prone to vanishing gradients problem.

3.2.2. Generator User Model

The Generator User Model tries to emulate a real users decision making mechanism. For example, given a set of items displayed to the user at a given time step (display set), the Generator User Model tries to take an action from the possible set of actions which would ideally be the action that a real user would have taken. The set of possible actions the model are the union of the items in the display set and the action of not choosing (e.g. clicking) on any of those items in the display set. This model is formulated as a Deep Neural Network with Fully connected layers and activation functions in between those layers. The inputs to the model are the state representation vector of the user (who we try to emulate the acitons of) and the possible actions (display set + not clicking on any item) at that time step. The display set consists of concatenated vecotrs of the one hot encodings vectors of the items in the display set. To this vector we concatenate another vector which represents not clicking on any of those items actions. Finally, we concatenate the state representation vector to this vector and feed into our Generator User Model. Note that the state representation vector is obtained from the history LSTM model. Finally, the model outputs a probability distribution over the possible actions which we greedly take the maximum to decide as the action that a user (fake) takes. It is important to note that in the GAN formulation, this Generator User Model acts as the Generator model. It generates the actions that a real user would take. This model aims to generate these actions so realisticly that it would fool the Discriminator Reward Model to give high utility values to its outputs, indicating that the Discriminator is fooled by the Generator User Model. In the original paper (1), they did not have a separate neural network architecture for the Generator User Model. Instead, they (1) took the generated action as the action which the Discriminator Reward Model gave the maximum reward to. The underlying idea behind this is that the Discriminator Reward Model generates the highest reward to the action that is mostly likely to be taken by a real user which is indeed the goal of the Generator User Model. We did not like this approach since, we felt like it took away from the mini-max game played between the Generator and the Discriminator by not having these units separately. Also, we wanted to experiment with having a separate Generator

User Model since we deemed it to be a better practice for us to have as part of the COMP547 course.

3.2.3. Discriminator Reward Model

The Discriminator Reward model aims to distinguish actions taken by a real user (real actions) from the actions generated by the Generator User Model (fake actions). To do so, the Discriminator Reward Model would take the user's state/history representation generated by the history LSTM along with the displayed set as it's input. Similar to the Generative User Model's inputs, we would concatenate the aforementioned vectors and feed them to the Discriminator Reward Model to generate rewards for each possible action. The generated rewards would ideally be higher for the actions that are more in accordance with the corresponding user (state representation vector), and lower for the actions that are unlikely to be taken by a real user therefore, be likely to be generated by the Generator User Model (fake actions). The Discriminator Reward Model is formulated as a Deep Neural Network with fully connected layers and activation functions between those layers. During the training, the Discriminator Reward Model would try to get good at distinguishing actions taken by a real user from the actions generated by the Generator User Model by predicting higher rewards for real user actions and lower actions for emulated user actions. The Model would be playing a mini-max game with the Generator User Model. In the original paper (1), the Discriminator Model's reward functions were used to generate actions in a greedy manner instead of having a separate network for the Generator User Model. In our work, the Discriminator Reward Model is completely separated from the Generator User Model.

3.3. Adversarial Training

Training loss formulation resembles a mini-max game between the Discriminator Reward Model and the Generator User Model. This generative adversarial training formulation can be seen in 1.

Generative Adversarial Training:

$$\min_{\theta} \max_{\alpha} \left(\mathbb{E}_{\phi_\alpha}\left[\sum_{t=1}^{T} r_\theta(\boldsymbol{s}^t_{true}, a^t)\right] - R(\phi_\alpha)/\eta\right) - \sum_{t=1}^{T} r_\theta(\boldsymbol{s}^t_{true}, a^t_{true}), \qquad (5)$$

Figure 1. Mini-max Loss Formulation

The loss function consists of three main parts namely, the reward for the ground truth (real user actions) data, the reward for the generated user actions, and the regularization term. The reward values here are generated by the Discriminator User Model. The reward values for the generated user actions are approximated using sampling which is represented by an Expectancy in the formula (1). The Generator User Model tries to generate actions which would yield high reward values by fooling the Discriminator Reward Model to think that the actions are good enough to be taken by a real user. On the other hand, the Discriminator Reward Model tries to give low reward values for the actions generated by the Generator User Model since, they are not coming from a real user. The ultimate goal of the training would be to have a user model (generator) which is generating actions as good as a real user so that the generator would have to make a guess (0.5 probability) about whether the action was taken by a real user or the action was emulated by the Generator User Model. The second part of the loss term concerns the ground truth (real user) actions. Given the ground truth actions the Discriminator Reward Model generates reward values. The goal is to have the reward function give high rewards for the actions taken by the real user. The aforementioned first term (generator loss term) and the second term (ground truth loss) is summarized by reward function trying to maximize the gap between the reward values for the generated user actions and the reward for the real user actions. Simultaneously, the Generator User Model is trying to minimize this gap by fooling the Discriminator Reward Model to think that the generated actions are coming from a real user.

3.3.1. Regularization

The last term is the regularization term. This term is used to stabilize the training of the generative adversarial training. This term depends on the actions generated by the user model (generator) therefore it impacts the way that the user model's parameters are updated. In the original paper (1), this term is also used to control the exploration/exploitation of the greedy user model (Taking the max over the Discriminator Reward Model's outputs to generate fake action). By shifting the weights of this term they were able to encourage the model to explore new states. In the original paper (1), the authors tried using Shannon entropy, and L2 regularization. By choosing a certain regularization term, the authors obtained a closed form solution of the general generative adversarial loss in 1. We implemented the general version of this formula (with no specific closed form solution) which would allow for experimentation with different regularization functions without making any further assumptions.

4. Experimental Results

We will compare our user model with the baselines suggested by (1). We were able to obtain identical train test splits for 3 of the datasets from the authors' Tensorflow 1 implementation. These datasets are: Yelp which contains user reviews and 9 nearest businesses as the display set, Taoboa which contains clicking and buying records data of users and

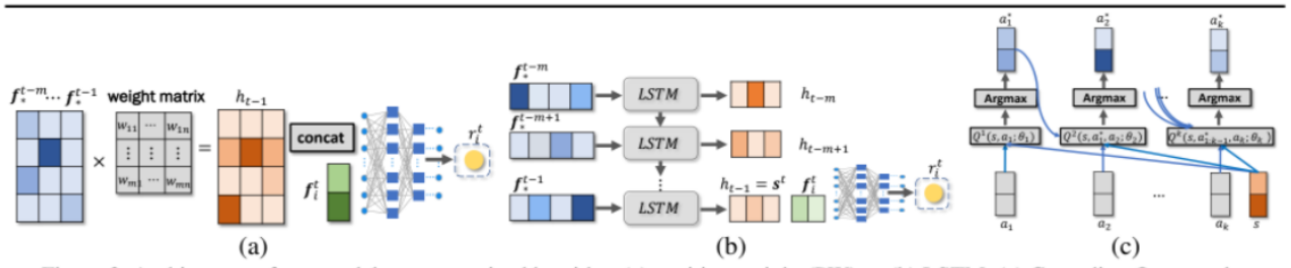


Figure 2. Architecture of our models parameterized by either (a) position weight (PW) or (b) LSTM. (c) Cascading Q-networks.

Figure 2.

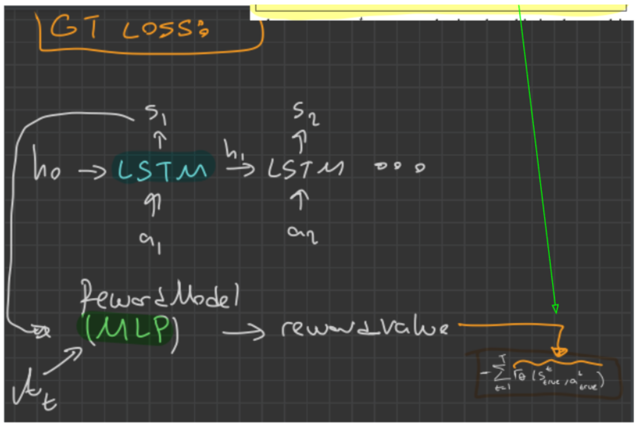


Figure 3. Ground Truth Loss Formulation

RecSys15YooChoose which contains click stream data that sometimes end with purchase events. Using these datasets and identical train-test splits we will be able to compare our model with the results of the baselines and proposed models of (1). Specifically we will use Top-k precision which will measure the ratio of actual clicks among th top-k rated items of the display set. This represents a greed agent, the simplest reinforcement learning approach. Base lines used are W&D-LR (3), CCF (8), IKNN (5), S-RNN (5), SCKKNNC (6), XGBOOST (2), DFM (4), SCKNNW (5) approaches which we mentioned with more detail i related works. We used the discriminator model directl to select the actions with the maximum rewards in orde to replicate(1)'s greedy action selection. However we als trained a separate generator network unlike (1). Therefor we used top-1 precision metric to compare our generato network with the greedy user model.

As it can be seen from the training loss curves Yelp, Rec Sys15YooChoose and Taobao datasets from figures 4,5 an 6 respectively, we failed to train a successful model. Thi is seen further with the validation loss curves from figure **??**,7,8 again corresponding to Yelp, RecSys15YooChoos and Taobao datasets respectively. The baseline top-k metrics from (1) can be seen in Figure 9. Our top-k results can be found in figures 10,11,12 corresponding to Yelp, RecSys15YooChoose and TaoBao datasets respectively. Our models did not get any meaningful scores as we expected from the training curves. We experimented with random initialization without training and got the scores in Figure 13.This random initialization almost matches some of the baselines.

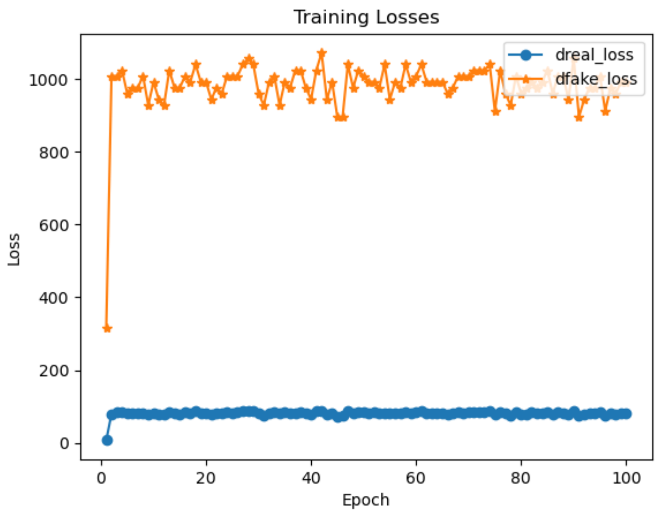


Figure 4. Yelp Training Loss Curve

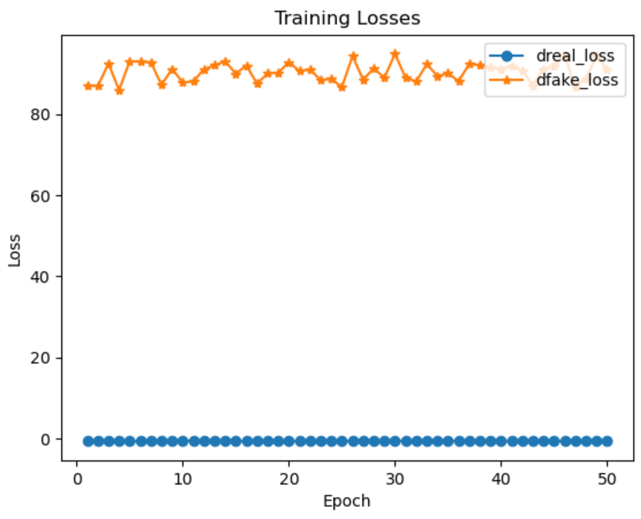


Figure 5. RecSys15YooChoose Training Loss Curve

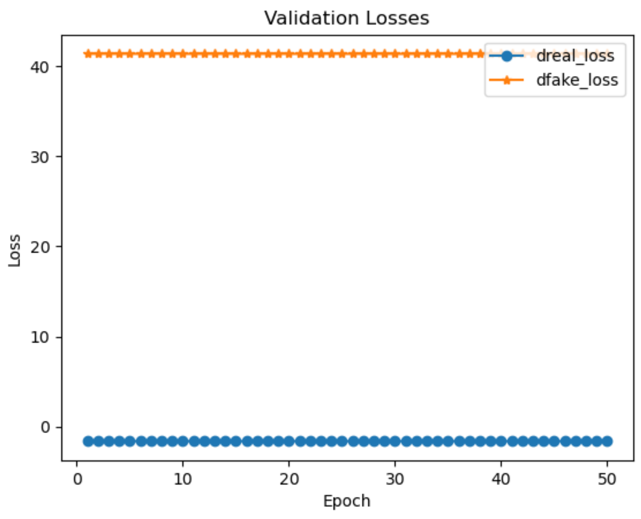


Figure 7. RecSys15YooChoose Validation Loss Curve

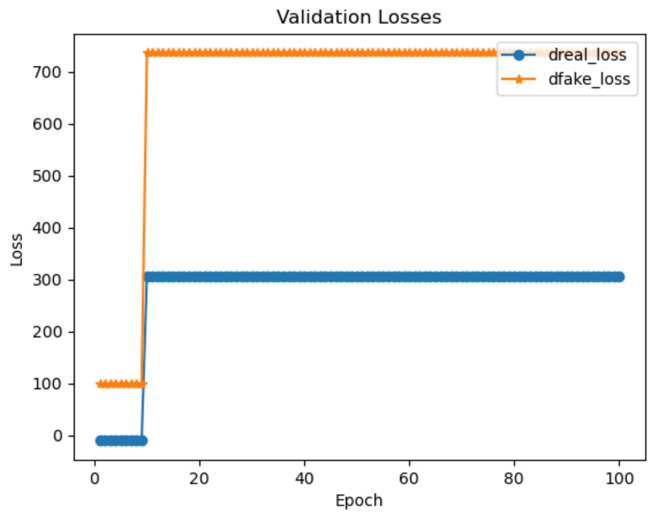


Figure 8. TaoBao Validation Loss Curve

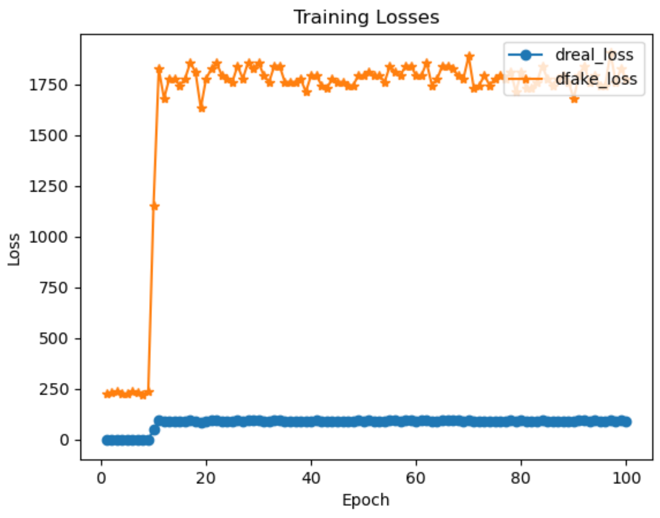
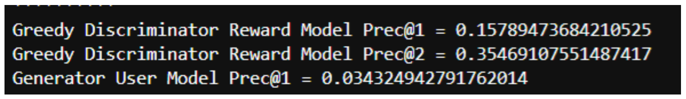
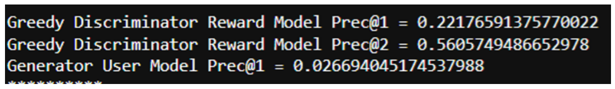


Figure 6. TaoBao Training Loss Curve



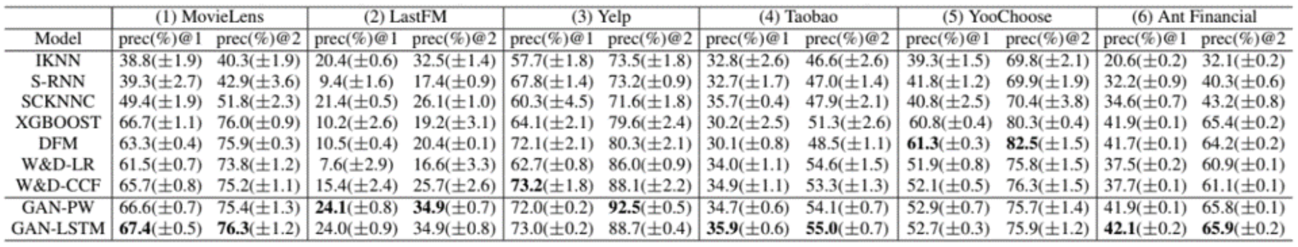
```
Greedy Discriminator Reward Model Prec@1 = 0.15789473684210525
Greedy Discriminator Reward Model Prec@2 = 0.35469107551487417
Generator User Model Prec@1 = 0.034324942791762014
```

Figure 10. Yelp Top-k Metrics



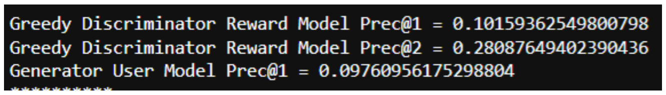
```
Greedy Discriminator Reward Model Prec@1 = 0.22176591375770022
Greedy Discriminator Reward Model Prec@2 = 0.5605749486652978
Generator User Model Prec@1 = 0.026694045174537988
```

Figure 11. RecSys15YooChoose Top-k Metrics



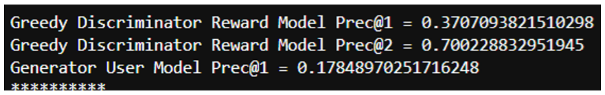
	(1) MovieLens		(2) LastFM		(3) Yelp		(4) Taobao		(5) YooChoose		(6) Ant Financial	
Model	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2	prec(%)@1	prec(%)@2
IKNN	38.8(±1.9)	40.3(±1.9)	20.4(±0.6)	32.5(±1.4)	57.7(±1.8)	73.5(±1.8)	32.8(±2.6)	46.6(±2.6)	39.3(±1.5)	69.8(±2.1)	20.6(±0.2)	32.1(±0.2)
S-RNN	39.3(±2.7)	42.9(±3.6)	9.4(±1.6)	17.4(±0.9)	67.8(±1.4)	73.2(±0.9)	32.7(±1.7)	47.0(±1.4)	41.8(±1.2)	69.9(±1.9)	32.2(±0.9)	40.3(±0.6)
SCKNNC	49.4(±1.9)	51.8(±2.3)	21.4(±0.5)	26.1(±1.0)	60.3(±4.5)	71.6(±1.8)	35.7(±0.4)	47.9(±2.1)	40.8(±2.5)	70.4(±3.8)	34.6(±0.7)	43.2(±0.8)
XGBOOST	66.7(±1.1)	76.0(±0.9)	10.2(±2.6)	19.2(±3.1)	64.1(±2.1)	79.6(±2.4)	30.2(±2.5)	51.3(±2.6)	60.8(±0.4)	80.3(±0.4)	41.9(±0.1)	65.4(±0.2)
DFM	63.3(±0.4)	75.9(±0.3)	10.5(±0.4)	20.4(±0.1)	72.1(±2.1)	80.3(±2.1)	30.1(±0.8)	48.5(±1.1)	**61.3**(±0.3)	**82.5**(±1.5)	41.7(±0.1)	64.2(±0.2)
W&D-LR	61.5(±0.7)	73.8(±1.2)	7.6(±2.9)	16.6(±3.3)	62.7(±0.8)	86.0(±0.9)	34.0(±1.1)	54.6(±1.5)	51.9(±0.8)	75.8(±1.5)	37.5(±0.2)	60.9(±0.1)
W&D-CCF	65.7(±0.8)	75.2(±1.1)	15.4(±2.4)	25.7(±2.6)	**73.2**(±1.8)	88.1(±2.2)	34.9(±1.1)	53.3(±1.3)	52.1(±0.5)	76.3(±1.5)	37.7(±0.1)	61.1(±0.1)
GAN-PW	66.6(±0.7)	75.4(±1.3)	**24.1**(±0.8)	**34.9**(±0.7)	72.0(±0.2)	**92.5**(±0.5)	34.7(±0.6)	54.1(±0.7)	52.9(±0.7)	75.7(±1.4)	41.9(±0.1)	65.8(±0.1)
GAN-LSTM	**67.4**(±0.5)	**76.3**(±1.2)	24.0(±0.9)	34.9(±0.8)	73.0(±0.2)	88.7(±0.4)	**35.9**(±0.6)	**55.0**(±0.7)	52.7(±0.3)	75.9(±1.2)	**42.1**(±0.2)	**65.9**(±0.2)

Figure 9. Baselines from (1)



```
Greedy Discriminator Reward Model Prec@1 = 0.10159362549800798
Greedy Discriminator Reward Model Prec@2 = 0.28087649402390436
Generator User Model Prec@1 = 0.09760956175298804
**********
```

Figure 12. TaoBao Top-k Metrics



```
Greedy Discriminator Reward Model Prec@1 = 0.3707093821510298
Greedy Discriminator Reward Model Prec@2 = 0.700228832951945
Generator User Model Prec@1 = 0.17848970251716248
**********
```

Figure 13. Yelp Random Initialization Top-k Metrics

5. Conclusions

We have two hypothesis to explain our inability to train a functioning model. First hypothesis is that the datasets that we used are sparse. Unlike (1) we choose to pad the sequence of user clicks to the longest sequence in order to be able to use batches with multiple users in training. This also required us to pad the display sets of each time step to the longest display set as well. Due to the padding and the already short sequences the dataset became too sparse. One-hot encoding the items might have also contributed to this problem. Our second hypothesis is that by adding a separate generator network to our model on top of the History LSTM and the discriminator we ended up with too complex of a model to optimize with the given datasets. In (1), there were mentions of pretraining the discriminator network before the GAN training to improve stability. We found this counter-intuitive since (1) did not have a separate generator network anyways. We decided to add the generator network to make this a true GAN model. Although we could not find the exact cause of the instability, we believe it is caused by a combination of the two hypothesis we mentioned and the instable nature of training GANs. Different regularizations, loss and GAN formulations might solve the issues that we encountered. However we did not have the time to test and find a solution as we spent the majority of our time with the dataset.

5.1. Acknowledgements

All of the efforts are shared it is not possible for us to divide the tasks done with a clear boundary. We both worked on the same tasks at the same time.

References

[1] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song, “Generative adversarial user model for reinforcement learning based recommendation system,” arXiv.org, 01-Jan-2020. [Online]. Available: https://arxiv.org/abs/1812.10613. [Accessed: 10-Apr-2022].

[2] Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794. ACM, 2016

[3] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. Wide & deep learning for recommender systems. In Proceedings of the 1st Workshop on Deep Learning for Recommender Systems. ACM, 2016.

[4] Guo, H., Tang, R., Ye, Y., Li, Z., and He, X. Deepfm: a factorization-machine based neural network for ctr prediction. arXiv preprint arXiv:1703.04247, 2017.

[5] Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. Session-based recommendations with recurrent neural networks. In ICLR, 2016.

[6] Jannach, D. and Ludewig, M. When recurrent neural networks meet the neighborhood for session-based recommendation. In Proceedings of the Eleventh ACM Conference on Recommender Systems, pp. 306–310. ACM, 2017.

[7] Li, L., Chu, W., Langford, J., and Schapire, R. E. A contextual-bandit approach to personalized news article

recommendation. In Proceedings of the 19th international conference on World wide web, pp. 661–670. ACM, 2010.

[8] Yang, S.-H., Long, B., Smola, A. J., Zha, H., and Zheng, Z. Collaborative competitive filtering: learning recommender using context of user choice. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, pp. 295–304. ACM, 2011.