A Practical Deep Online Ranking System in E-commerce Recommendation

Yan Yan[1], Zitao Liu[2], Meng Zhao[1], Wentao Guo[1], Weipeng P. Yan[1], and Yongjun Bao[1]

[1] Intelligent Advertising Lab, JD.COM,
675 E Middlefield Rd, Mountain View, CA USA
{yan.yan, zhaomeng1, guowentao, paul.yan, baoyongjun}@jd.com
[2] TAL AI Lab, TAL Education Group, Beijing, China
liuzitao@100tal.com

Abstract. User online shopping experience in modern e-commerce websites critically relies on real-time personalized recommendations. However, building a productionized recommender system still remains challenging due to a massive collection of items, a huge number of online users, and requirements for recommendations to be responsive to user actions. In this work, we present our relevant, responsive, and scalable deep online ranking system (DORS) that we developed and deployed in our company. DORS is implemented in a three-level architecture which includes (1) candidate retrieval that retrieves a board set of candidates with various business rules enforced; (2) deep neural network ranking model that takes advantage of available user and item specific features and their interactions; (3) multi-arm bandits based online re-ranking that dynamically takes user real-time feedback and re-ranks the final recommended items in scale. Given a user as a query, DORS is able to precisely capture users' real-time purchasing intents and help users reach to product purchases. Both offline and online experimental results show that DORS provides more personalized online ranking results and makes more revenue.

Keywords: Recommender system · E-commerce; Deep learning · Multi-arm bandits.

1 Introduction

Building a relevant and responsive recommender system is the key to the success of e-commerce and online retailing companies. A desired recommender system helps people discover things they love and potentially purchase them, which not only makes companies profitable but also brings convenient online shopping experience to people's daily lives.

There is a huge amount of data generated on e-commerce websites everyday, such as user historical online behaviors, product inventory specifics, purchase transactions, etc. However, how to utilize them and build a practical recommender system that recommends billions of items to millions of daily active

users still remains challenging due to the *relevance* and *responsiveness* trade-offs arising from online shopping scenarios. Briefly, when a recommender system is designed to serve the most relevant items, it uses every piece of available information about the users, such as demographical information, past shopping habits, favorite and disliked items, etc., to score, rank and select recommendation candidates. Because of the tremendous dimensions of information and the huge size of candidates, the entire recommendation process is time consuming and cannot be finished in real time [3]. Hence, the recommended items become stale to the most recent shopping interests of users. On the other hand, a good recommender system needs to be responsive and is able to capture user shopping intent in real time. User shopping intents drift quite a lot and users tend to shop in different categories even within a single session [4]. Responsiveness is difficult to achieve since given a limited time window (request time out window), only a few features can be used to score and rank the results. The majority of existing approaches proposed for e-commerce recommendation in the literature only focus on either improving recommendation relevance or achieving responsiveness in a computationally expensive approach which is not affordable or applicable when serving thousands of recommendation requests every second.

In this work, we address this problem by presenting our *D*eep *O*nline *R*anking *S*ystem, i.e., *DORS* which is relevant, responsive and scalable and is able to serve millions of recommendation requests everyday. DORS is designed and implemented in a three-level novel architecture, which includes (1) candidate retrieval; (2) learning-to-rank deep neural network (DNN) ranking; and (3) online re-ranking via multi-arm bandits (MAB). The candidate retrieval stage enables us to incorporate various business rules to filter out irrelevant items, such as out-of-stock items, etc. The learning-to-rank DNN stage fully utilizes all available information about users, items and their corresponding interaction features and conducts a full scoring, which provides a fine-grained high-recall set of candidates. The last stage of online re-ranking takes the output of DNN top K ranking results and incorporates user real-time online feedback, such as impressions, clicks, purchases, etc. to adjust the final recommendations.

Overall this paper makes the following contributions:

- It presents a practical three-stage recommendation system that is able to serve relevant and responsive recommendations to millions of users.
- It provides a robust and flexible production system that is easy to implement and is superior in terms of computational efficiency and system latency.
- We evaluate our approach and benefits in our real-world scenarios with millions of real traffics.

2 Related Work

Various methodologies have been built by machine learning and data mining community to conduct better recommendations in different scenarios [21,35,14],

[3] Real time is defined as under 200 milliseconds.
[4] Session is defined as a 30-minute window in this paper.

which can be divided into the following categories: static recommendation (Section 2.1), time-aware recommendation (Section 2.2) and online recommendation (Section 2.3).

2.1 Static Recommendation

Static recommendation makes the assumption that both the users and items rarely change over time and it tries to optimize the relatedness between users and items given the entire history. All standard and classic recommendation algorithms such as content based recommendations [18,20], user or item based collaborative filtering [26], non-negative matrix factorization [24,36] fall into this category. Such approaches are effective and serve as the core services in lots of Internet companies [9,13,16]. However, the recommender system by nature is a dynamic process, which means user online actions or events happened in a sequential manner and user shopping interests drift over time. Moreover, with the development of fast and easy Internet access, more frequent online user behaviors are observed in real time (under 500 millisecond). It is challenging or even inapplicable for static recommendation approaches to be responsive and get updated once user feedback is observed.

2.2 Time-aware Recommendation

Besides modeling the relations between users and items, various researches have been done by taking *time* as the third dimension to capture the evolution of user interest. Time-aware recommendation approaches aim at explicitly modeling user interests over time slices [32,15,29,28] and they can be combined with various classic recommendation algorithms in static recommendation (Section 2.1). For examples, Ding and Li introduced a personalized decay factor according to purchase behaviors and proposed time weight collaborative filtering [11]. Yehuda developed a collaborative filtering with temporal dynamics which each prediction is composed of a static average value and a dynamic changing factor [17]. Yin et al. proposed a user-tag-specific temporal interests model for tracking users' interests over time by maximizing the time weighted data likelihood [34]. Liang et al. used a tensor factorization approach to model temporal factors, where the latent factors are under Markovian assumption [33]. Lu et al. improved the standard low-rank matrix factorization by using Kalman filter to infer changes of user and item factors [22]. Gao et al. modeled the correlations between user check-in timestamps and locations through an improved matrix factorization approach [12]. Although many studies above have been considered the temporal aspects of data, most of them are far away from being responsive. The majority of proposed approaches model the temporal dependence by a pre-defined time window, such as hours, days, or weeks, which is far away from serving real-time recommendations.

2.3 Online Recommendation

To capture user real-time feedback, several studies focus on improving online learning algorithms so that recommendation models get updated immediately once the user online actions happen [7,1,25,10,6]. Agarwal et al. developed an online bilinear factor model to learn item-specific factors through online regression. The online regression for each item can be performed independently and hence the procedure is fast, scalable and easily parallelizable [1]. Rendle et al. derived an online-update algorithm for regularized kernel matrix factorization models that allows to solve the cold start problem [25]. Diaz et al. presented a stream ranking matrix factorization technique, which optimizes the personalized ranking of topics. They also proposed a selective sampling strategy to perform online model updates based on active learning principles, that closely simulates the task of identifying relevant items from a pool of mostly uninteresting ones [10]. Chang et al. viewed the online recommendation problems as an explicit continuous-time random process, which generates user interested topics over time. They also provided a variational Bayesian approach to permit instantaneous online inference [6]. However, even the majority work above presents approaches to conduct parallelized online updates with user feedback there are two drawbacks. First, recommendation model parameters tend to be resistant to a single user feedback online update and hence it is not sufficient to provide personalized results. Second, the majority work mentioned above doesn't scale well when building a recommender system to suggest billions of items to millions of daily active users.

3 The Deep Online Ranking System

In this work, we develop a three-level recommender system architecture that is able to flexibly provide the most relevant and responsive recommendation to our millions of daily active users. The entire recommendation workflow in our system includes three phases: (1) candidate retrieval that scans all available item inventory by using forward and backward indices and trims out candidates violating the business rules (Section 3.1); (2) full features scoring and ranking via DNN that uses the pre-trained DNN to rank all candidates in order to optimize the gross merchandise volume (Section 3.2); (3) online re-ranking via MAB that dynamically adjusts the ranking results from DNN by using a novel multi-arm bandits algorithm. It takes user real-time feedback such as clicks and captures user shopping intents as fast as possible (Section 3.3). The entire workflow is illustrated in Figure 1.

3.1 Candidate Retrieval

Candidate retrieval serves as the first phase of our recommendation system and it is triggered when a user request is received. The recommender system fetches both user historical data and item features. Then all these information are passed

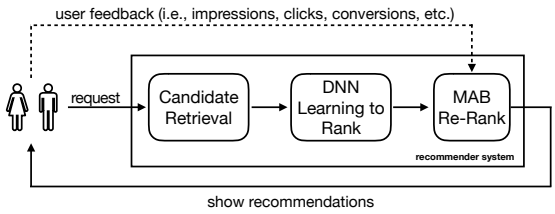

Fig. 1. Workflow overview of our recommender system.

to our retrieval system to select initial candidates for full feature scoring. Furthermore, business rule based item trimming or filtering are also done in candidate retrieval phase. In a productionized e-commerce recommender system, various business rules have to be enforced, such as out-of-stock items shouldn't be retrieved, the number of items from the same brand should be limited, etc.

3.2 Learning-to-rank via DNN

In the ranking stage, we develop our novel *learning-to-rank* DNN model to score all candidates returned from the retrieval stage by using full sets of features and hence generate a fine-grained set of items $\mathcal{S}$. Generally speaking, this is achieved by sophisticated machine learning models such as factorization machines [23], pairwise *learning-to-rank* [4,27], listwise *learning-to-rank* [5], etc. In this work, the ranking module is implemented by a pairwise *learning-to-rank* DNN.

The Ranking Problem Let $\boldsymbol{x}_i$ be the m-dimensional feature vector of item i. Each item i belongs to its category c_i. A category in e-commerce generally represents a group of products that share the same utilities or have similar functionalities: *e.g.* apparels, cellphones, snacks, etc. Each item i also associates with its *gross merchandise volume* (GMV) gmv_i which represents the product's retail value and it in most cases corresponds to the item's final price.

In this work, the goal is that given all retrieved items, select the top K items so that the discounted cumulative gain (DCG) of GMV is optimized and the DCG_K is defined as follows $DCG_K = \sum_{s_i \in S} gmv_{s_i} \mathcal{I}(s_i) / \log_2(i+1)$ and $\mathcal{I}(s_i)$ is the indicator function such that $\mathcal{I}(s_i) = 1$ when s_i is purchased and otherwise, $\mathcal{I}(s_i) = 0$.[5]

Learning In order to optimize the DCG objective, we develop a *learning-to-rank* DNN model [3,8]. More specifically, our DNN model is implemented by a

[5] The motivation for optimizing GMV is related to the e-commerce company's business model. Since investors and shareholders utilize the settled GMV from all e-commerce transactions generated to estimate the current health and future potential of the corresponding company, maximizing GMV becomes critical. In general, if each item's GMV sets to be **1**, optimizing GMV degrades to optimizing the sales conversion.

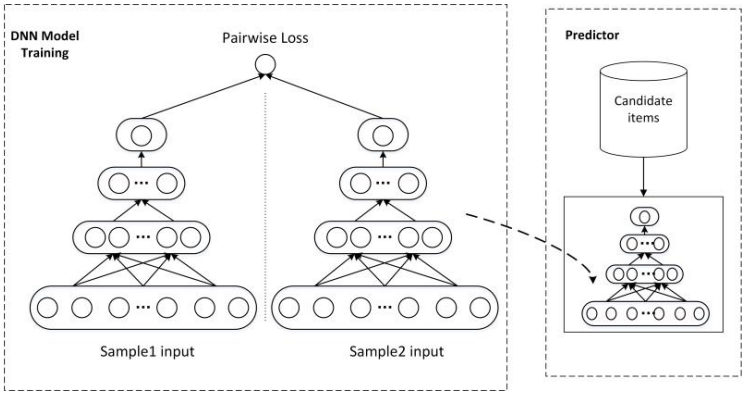

Fig. 2. Pairwise architecture of our learning-to-rank DNN.

pair of 5-layer DNNs that share the same set of model parameters. Each DNN includes 1 input layer, 3 fully-connected layers and 1 output layer. The input layer takes feature $\mathbf{x}_i$, which contains both item and user information via one-hot encoding. The high level model structure is shown in Figure 2 (Left).

In our training, instead of using the binary labels which indicate whether items are bought, we conduct calibrations on binary labels and convert them into real-valued labels y_is, where $0 \leq y_i \leq 1$. The calibrated labels (y_is) are computed by using item s_i's GMV gmv_{s_i} that represents how much revenue will be generated if s_i is impressed comparing against the maximum gmv under the same category c_{s_i}, shown in eq.(1).

$$y_i = \mathcal{Y}(gmv_{s_i}, \mathcal{I}(s_i)) = \frac{gmv_{s_i}}{\max_{s \in c_{s_i}} gmv_s} \times \mathcal{I}(s_i) \tag{1}$$

To train our pairwise DNN model (shown in Figure 2), we generate training example pairs by combining a positively scored item and a randomly selected zero scored item ($y = 0.0$). Furthermore, we define the loss function of the proposed DNN as follows.

$$\mathcal{L} = \sum_j [(\hat{y}_{1j} - y_{1j})^2 + (\hat{y}_{2j} - y_{2j})^2 + \lambda \max(0, \gamma - (\hat{y}_{1j} - \hat{y}_{2j})(y_{1j} - y_{2j}))] \tag{2}$$

where $< \mathbf{x}_{1j}, y_{1j} >$ and $< \mathbf{x}_{2j}, y_{2j} >$ are the first and second examples in the jth pair. $\hat{y}_{1j}$ and $\hat{y}_{2j}$ are the outputs of the DNN predictions of the first and second sample: $\hat{y} = \mathcal{DNN}(\mathbf{x})$. λ is the weighting parameter that balances the square loss of each prediction and the hinge loss of the prediction distance for such pair. γ serves as the classification margin to make sure that separability between each pair is preferred.

Prediction In the prediction phase, each item $\mathbf{x}_i \in \mathcal{S}$ passes through the well trained *learning-to-rank* DNN predictor (Figure 2 (Right)), and DNN predicts the output score $\hat{y}_i$, i.e., $\hat{y}_i = \mathcal{DNN}(\mathbf{x}_i)$. $\hat{y}_i$ is later served as the candidate score of item $\mathbf{x}_i$ for the online MAB based ranker to make further re-ranking.

3.3 Online re-ranking via MAB

Even the pre-trained DNN model in Section 3.2 is good at selecting items, it fully relies on the historical aggregated user information, which tends to be outdated and cannot reflect the up-to-date user intents. Furthermore, user shopping intents vary with time and they change even within a single session. For example, tens or hundreds of impressions without a single click may indicate users are bored of current recommendations and it may be worthwhile to make "broader" recommendation and let users explore products in other categories. On the other hand, if a few clicks or purchases are observed, the following recommendations probably need to be "focused" and allow users to fully exploit relevant products. It is necessary to have such responsive recommender systems that keep track of transient shopping intentions while users are browsing the product feeds.

It is very difficult to achieve such responsiveness by using the DNN model in the second phase (Section 3.2) since doing full scoring of all the candidates after each user click or purchase event is computationally prohibitive. Moreover, it is not easy to have an online update to incorporate most recent events into the ranking model and hence the ranking model may become stale.

In this work, we develop a dynamic online re-ranking algorithm by revising the classic Thompson sampling. We pick Thompson sampling out of other MAB approaches due to its implementation straightforwardness. Further, by careful revising, the *revised*-Thompson sampling turns out to be superior in terms of performance metrics which we will elaborate in more detail. Our MAB based re-ranking algorithm is able to

- promote recommended items that users are potentially interested in;
- demote items that users are intentionally ignored;
- explore items from different categories to diversify the ranking results.

Here, we follow the problem settings of the contextual multi-arm bandits problem in [19], and define the contextual bandits (see Definition 1) and rewards (see Definition 2) in DORS as follows.

Definition 1. *(*Contextual Bandits in DORS*) Let each arm represent a product category c and $\mathcal{S}$ be a set of K top ranked items from DNN. Such K items are then divided into different arms based on their own product categories. The player pulls one arm c_i at round i, selects the top item $\mathbf{x}_i$ from the chosen arm which has not been displayed yet and contiguously places the item to the item selected from round $i-1$. The reward at round i is observed as $gmv_i\mathcal{I}(\mathbf{x}_i)$. Ideally, we would like to choose arms such that the total rewards are maximized.*

Definition 2. *(*Rewards in DORS*) The rewards are defined as the total GMV generated from $\mathcal{S}$ that users place orders on. In general it can be written as:* $Revenue(K, \mathcal{S}) = \sum_{s_i \in \mathcal{S}} gmv_{s_i} \mathcal{I}(s_i)$.

The *revised*-Thompson sampling algorithm is triggered after *learning-to-rank* DNN. After taking the DNN output as the static ranking results, DORS organizes items by categories, and fine tunes the local orders of the DNN static ranking, which enables more efficient convergence by enforcing DNN scores as the warm starts and analyzing user feedback including impressions and clicks as rewards signals for the Thompson online re-ranking.

At the initialization stage, we group the *pre*-ranked items based on their categories, and define each group as a single arm, which is modeled by a unique beta distribution as follows

$$f(r; \alpha_c, \beta_c) = \frac{\Gamma(\alpha_c + \beta_c)}{\Gamma(\alpha_c)\Gamma(\beta_c)} r^{\alpha_c - 1}(1-r)^{\beta_c - 1} \tag{3}$$

where α_c and β_c are two hyper-parameters.

Let avg_c represent the average DNN scores of all items in category c. We initialize α_c, β_c, avg_c by DNN *pre*-ranked scores which represent how likely the user be interested in those items based on recent history. At round i, the *revised*-Thompson sampling randomly draws M samples $\{r\}_M$ based on M estimated beta distributions, and then selects the item associated with the highest adjusted score $\hat{y}_i$ for exposure at round i. If the item is clicked, the algorithm updates α_{c_i} in arm c_i. Otherwise, the algorithm updates β_{c_i} in arm c_i. The details of the MAB algorithm in DORS are described in Algorithm 1. Please note that the values of θ_1, θ_2, θ_3, δ in our current production are selected by grid searches in online A/B experiments.

One challenge that stops people from using the multi-arm bandits model to the real production system might be the lack of existence of reseaches regarding the multi-arm bandits convergence analysis. Such convergence efficiency is important since most customers in average only browse less than one handred items *per* visit. In this section, we prove that the *revised*-Thompson sampling in DORS guarantees the efficient convergence and provide the regret boundaries in two cases.

As described in earlier sections, arms are pulled based on the beta distributions $f(r; \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1}(1-r)^{\beta-1}$, and for each item, the rewards are set to be $gmv_{s_i}\mathcal{I}(s_i)$, indicating whether a gmv related user action would happen at the current round. Without loss of generality, by assuming each item's gmv identity and taking the expectation, we simplify the total rewards and translate it into the modified total expected regrets as:

$$\mathbb{E}(Regret(K, \mathcal{S})) = \mathbb{E}[\sum_{k=1}^{K}(\mu^* - \mu_i(k))] = \sum_i \Delta_i \mathbb{E}[h_i(K)] \tag{4}$$

where $\Delta_i = \mu^* - \mu_i(k)$ denotes the regret for pulling the i^{th} arm and h_i denotes the number of plays for i^{th} arm up to round $k-1$, μ^* denotes the maximum

reward (calibrated revenue) the system could possibly reach for exposing item i in round i, here in our case $\mu^* = 1.0$. Suggested in Algorithm 1, unlike traditional Thompson sampling updating strategies, we utilized DNN score $\hat{y}$s to update hyper parameter α and β for each beta distribution with smoothing factors under our GMV identity assumptions.

Algorithm 1 MAB in DORS: the *revised*-Thompson sampling.

INPUT:
δ: control the intensity of negative feedbacks
$\theta_1,\ \theta_2,\ \theta_3$: control how much the $\mathcal{DNN}$ scores to be tuned
α_c, β_c: the hyper parameter of beta distribution for category c.
$\mathcal{S}$: the items that are top ranked by the $\mathcal{DNN}$ algorithm
$\mathcal{U}_c$: the items that are not selected in category c.
$\mathcal{E}_c$: the items that are impressed but not clicked in category c.
$\mathcal{A}_c$: the items that are impressed and clicked in category c.
M: the total number of categories / arms
$|\cdot|_0$: the cardinality operator.
procedure INITIALIZATION
  for each $\langle \mathbf{x}, \hat{y} \rangle \in \mathcal{S}$ **do**
    for arm c such that $\mathbf{x} \in c$ **do**
      $\alpha_c \ = \ \alpha_c + \hat{y}$
      $\beta_c \ = \ \beta_c + (1 - \hat{y})$
      $\mathcal{U}_c = \mathcal{U}_c \cup \{\langle \mathbf{x}, \hat{y} \rangle\}$
    $avg_c \ = \ \alpha_c / |\mathcal{U}_c|_0$
procedure AT ROUND-i MAB RANKING
  PULLING ARMS:
  for each arm c **do**
    sample $r \ \sim \ f(r; \alpha_c, \beta_c)$
    update all $\hat{y} \ = \ \hat{y} \times (1 + r/\theta_1)$ for $\langle \mathbf{x}, y \rangle \in c$
  pick category $c = \arg\max_c \{r_1,\ r_2, \ldots, r_M\}$
  pick item $\langle \mathbf{x}_i, \hat{y}_i \rangle = \arg\max_i \{\hat{y} \in c\}$
  $\mathcal{U}_c = \mathcal{U}_{c_i} - \langle \mathbf{x}_i, \hat{y}_i \rangle$
  FEEDBACK:
  if $\langle \mathbf{x}_i, \hat{y}_i \rangle$ is impressed but not clicked **then**
    $\mathcal{E}_c = \mathcal{E}_c \cup \{\langle \mathbf{x}_i, \hat{y}_i \rangle\}$
    $\beta_{c_i} \ = \ \beta_{c_i} + (1 - avg_{c_i}) \times (1 - \exp(-\frac{|\mathcal{E}_{c_i}|_0}{\delta})) \times \theta_2$
  if $\langle \mathbf{x}_i, \hat{y}_i \rangle$ is impressed and clicked **then**
    $\mathcal{A}_{c_i} = \mathcal{A}_{c_i} \cup \{\langle \mathbf{x}_i, \hat{y}_i \rangle\}$
    $\alpha_{c_i} \ = \ \alpha_{c_i} + avg_{c_i} \times (\frac{|\mathcal{A}_{c_i}|_0}{|\mathcal{E}_{c_i}|_0}) \times \theta_3$

Theorem 1. *For the 2-armed case the revised-Thompson sampling holds the expected regret:*

$$\mathbb{E}(Regret(K, \mathcal{S})) \leq \frac{40 \ln K}{\Delta} + \frac{48}{\Delta^3} + 18\Delta = \mathcal{O}(\frac{\ln K}{\Delta} + \frac{1}{\Delta^3}) \tag{5}$$

where Δ indicates the difference in mean between the reward from the optimal arm μ^* and the reward from the suboptimal arm μ_{sub}: $\Delta = \mu^* - \mu_{sub}$.

Theorem 2. *For the M-armed case ($M > 2$) the revised-Thompson sampling holds the expected regret:*

$$\mathbb{E}(Regret(K, \mathcal{S})) \leq \mathcal{O}(\frac{\Delta_{\max}}{\Delta^3_{\min}}(\sum_{a=2}^{M} \frac{1}{\Delta_a^2}) \ln K) \tag{6}$$

Detailed proof can be found in supplemental materials. With convergence guarantees, we know that the *revised*-Thompson sampling helps the system achieve the overall optimized GMV performance with the expected regret no greater than $\mathcal{O}(\frac{\Delta_{\max}}{\Delta^3_{\min}}(\sum_{a=2}^{M} \frac{1}{\Delta_a^2}) \ln K)$.

4 Experiments

4.1 Case Study

We first walk through a simple case study to evaluate different bandit algorithms under the streaming recommendation use cases. We pick three *state-of-the-art* bandit algorithms: ϵ-greedy [31], Upper Confidence Bound (UCB) [2], and Thompson sampling [30]. Specifically, we simulate two versions of Thompson sampling:

- ***revised*-Thompson**: the revised Thompson sampling with *learning-to-rank* DNN initializations (Algorithm 1);
- ***normal*-Thompson**: the normal Thompson sampling without initializations.

  The random selection is also evaluated as a naïve baseline.

In the simulation, we design $M = 5$ arms. If the user clicks, we set each item's reward as 1, and 0 otherwise. The way we simulate "click"s is by presetting a thresholding probability τ. When an item is selected by different algorithms at each round, we sample another probability f_{item} based on that item's real unknown beta distribution. If $f_{\text{item}} \geq \tau$, we assume the "click" happens; otherwise we assume the user is not interested in the item which results in "no click" at that round.

We run this simulation 10 times and at each time we operate $10,000$ rounds. The average performance is shown in Figure 3 and Figure 4. The left subfigures are the cumulative gains / regrets for different methods and the right subfigures zoom in the early stage of different algorithms' performance. As shown, ϵ-greedy remains suboptimal regarding both rewards and regrets; UCB and *normal*-Thompson perform almost equally well; while the *revised*-Thompson performs the best in terms of faster convergence comparing to UCB and *normal*-Thompson. This is because the *revised*-Thompson initializes its arms based on the DNN scores of each item and updates its hyper-parameters via the user

feedback. Hence, the *revised*-Thompson converges in less steps relative to other standard MAB approaches. The random selection no-surprisingly performs the worst against other approaches. Regarding with implementation, the *revised*-Thompson is straightforward and the overall system latency remains very low (the details are reported in Section 4.5). From this case study, the *revised*-Thompson proves itself to be the best choice out of many other MAB approaches for the online re-ranking module in DORS.

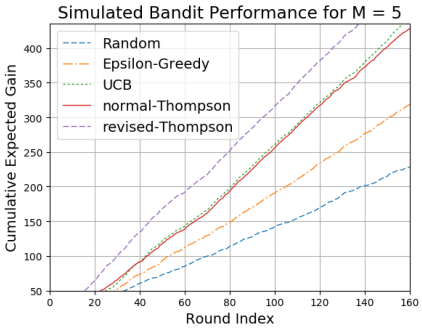

Fig. 3. MAB rewards simulation.

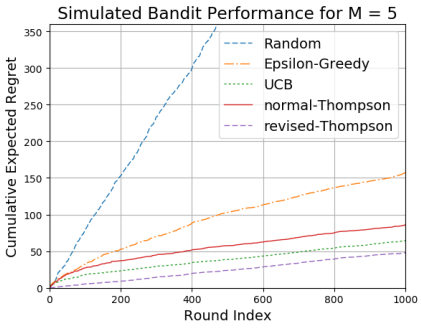

Fig. 4. MAB regrets simulation.

4.2 Experiment Setup

We conduct large-scale experiments on `www.jd.com`, which processes millions of requests in a daily basis. The total traffics have been divided into 11 buckets: the biggest bucket serves as the control bucket which takes about 20% of the total traffics and the rest 10 buckets gets about 8% traffics each, each of which is served as one testing bucket.

We first deploy DORS to one testing bucket for one week, and track the following metrics: GMV, order numbers, overall and page-wise (eq.(7)) normalized discounted cumulative gains ($NDCG$).

$$
\begin{aligned}
DCG_{p,k} &= \sum_{i=1,k\in\text{page}}^{p} \frac{gmv_{ki}\mathcal{I}(\mathbf{x}_{ki})}{\log_2(i+1)} \\
IDCG_{p,k} &= \max_{\mathcal{H}} DCG_{p,k} \\
NDCG_{p,k} &= DCG_{p,k}/IDCG_{p,k}
\end{aligned}
$$

$$
\begin{aligned}
&\Delta_{NDCG_{p,k}} = (NDCG_{p,k}^{\texttt{test}}/NDCG_{p,k}^{\texttt{control}} - 1.0) \times 100.0\% \qquad (7) \\
&\mathcal{H} : \text{all possible arrangement of items in page} - k
\end{aligned}
$$

Since the item lists are presented to users in a page-wise fashion and each page contains 4 - 20 items (front-page typically contains 8 items *per* page), page-wise $NDCG_p$ ($p = 8$) is a perfect metric for evaluating how DORS is performing in real world and how much gains observed are due to DORS online re-ranking.

Experiments include four methods and we briefly explain each of them as follows:

- **DORS**: our proposed online ranking framework which is composed by *learning-to-rank* DNN and the *revised*-Thompson sampling as online re-ranking;
- **DNN-*normal*-Thompson**: a right straight combination of *learning-to-rank* DNN and normal Thompson sampling without the specialized initialization. This model explains how the non-contextual bandits algorithm performs as the online ranker in the production system;
- **MAB-only**: is implemented via a normal Thompson sampling without *learning-to-rank* DNN and using the historical click through rate to update the beta distributions;
- **DNN-*rt-ft***: utilizes another *learning-to-rank* DNN with the same model structure. The difference is that besides utilizing offline features used in DORS, DNN-*rt-ft* takes most recent user feedback into the DNN training process to generate its own ranking results.

We report the performance of 7-day average page-wise $NDCG$ gain in Figure 5. At the first glance, it is clear that MAB-only performs the worst among all four algorithms, which explains the importance of the *learning-to-rank* DNN serving as the static ranking phase. Further, similar to what has been observed in the previous case study, without the specialized initialization, DNN-*normal*-Thompson fails to DNN-*rt-ft* due to the fact that by limited online signals, the *normal*-Thompson is slow in convergence. By our design, the proposed DORS beats the production baseline DNN-*rt-ft* by efficiently learning the user online intents.

4.3 Production Performance

Taking a closer look, we evaluate the DORS page-wise $NDCG$ percentage gains over DNN-*rt-ft* in Table 1 & Figure 6. By understanding each user recent behaviors via the personalized initialization and learning the real-time signals for online ranking, although they are not quite visible at page-1 (+1.47%), the gains for DORS at page-2 (+9.96%) and page-3 (+8.90%) quickly boost up, and then gradually diminish along with more items users browse. In the end, the gap between DORS and DNN-*rt-ft* becomes closer again at page-7 (+1.54%) and page-8 (+1.34%).

In terms of overall ranking performance, we report the final ranking NDCG percentage gain between DORS and DNN-*rt-ft* in a daily basis as shown in Table 2, as one can see, DORS consistently beats DNN-*rt-ft*. In average DORS's overall NDCG is 4.37% better comparing against DNN-*rt-ft*.

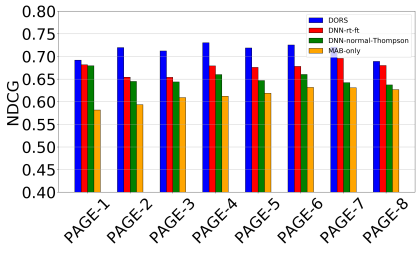

Fig. 5. 7-day page-wise NDCG for DORS, DNN-*rt-ft*, DNN-*normal*-Thompson, MAB-only.

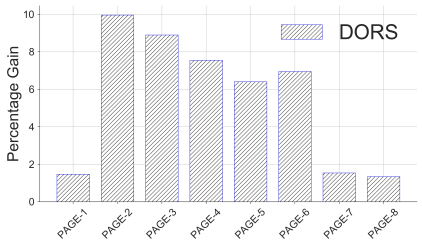

Fig. 6. 7-day page-wise NDCG percentage gain for DORS v.s. DNN-*rt-ft*.

We also report the daily GMV gain / loss for DORS, DNN-*normal*-Thompson and MAB-only over DNN-*rt-ft*. In average we see DORS has increased **16.69**% over DNN-*rt-ft* (Table 3). Daily GMV gains that are greater than 1.0% are typically translating into revenue in hundreds of thousands of dollars. On the one hand, DORS has proved its superiority against the current production DNN-*rt-ft* ranking algorithm in terms of operating revenue; on the other hand, without decent *learning-to-rank* DNN static ranking or carefully designed initializations, DNN + *normal*-Thompson (Table 4) and MAB-only (Table 5) both failed to beat the baseline by losing 8.08% and 19.57% in revenue respectively.

Page	Page-1	Page-2	Page-3	Page-4	Page-5	Page-6	Page-7	Page-8
Δ_{NDCG}	+1.47%	+9.96%	+8.90%	+7.55%	+6.40%	+6.95%	+1.54%	+1.34%

Table 1. Page-wise NDCG gain A/B test comparison: DORS v.s. DNN-*rt-ft*.

Date	Day1	Day2	Day3	Day4	Day5	Day6	Day7	**Average**
Δ_{NDCG}	+5.60%	+10.23%	+2.90%	+1.93%	+1.59%	+6.96%	+2.34%	**+4.37%**

Table 2. Daily NDCG gain A/B test comparison: DORS v.s. DNN-*rt-ft*.

Date	Day1	Day2	Day3	Day4	Day5	Day6	Day7	**Summary**
GMV	+22.87%	+45.45%	+20.20%	+2.73%	+0.91%	+23.15%	+1.50%	**+16.69%**
Orders	+2.14%	+1.57%	+5.18%	+0.42%	+2.79%	+4.19%	+2.20%	**+2.64%**

Table 3. GMV and orders comparison for DORS v.s. DNN-*rt-ft*.

Date	Day1	Day2	Day3	Day4	Day5	Day6	Day7	**Summary**
GMV	-12.08%	-9.33%	-4.74%	-3.24%	-18.31%	-7.49%	-1.43%	**-8.08%**
Orders	0.30%	-4.72%	-1.34%	-0.67%	-10.67%	-4.69%	-0.81%	**-3.23%**

Table 4. GMV and orders comparison for DNN + *normal*-Thompson v.s. DNN-*rt-ft*.

Date	Day1	Day2	Day3	Day4	Day5	Day6	Day7	**Summary**
GMV	-29.52%	-17.46%	-37.17%	-8.99%	-32.61%	-5.21%	-6.04%	**-19.57%**
Orders	-8.33%	-12.51%	-7.66%	-5.91%	-15.35%	-7.64%	-3.87%	**-8.75%**

Table 5. GMV and orders comparison for MAB-only v.s. DNN-*rt-ft*.

4.4 Distribution Analysis

It is worth analyzing the page-wise click distribution in a daily basis between DORS and DNN-*rt-ft* to better understand why the GMV has been significantly driven. Figure 7 is the daily page-wise click distributions over 7 days. For the figure readability, we display y-axis in the $\log_{10}$ scale, keep three significant digits and round up all numbers that are smaller than 1.0 to 1.0 ($\log_{10}(1.0) = 0.0$). Since each page displays 8 items for recommendations, the page-wise clicks could at most reach 8. The x-axis indicates the number of clicks people make and y-axis indicates the number of people making certain clicks at that page.

At page-1, the DORS histogram resembles the DNN-*rt-ft* histogram. This is due to the fact that the online user signals have not been fed into the MAB yet, so DORS is not expected to behave differently from DNN-*rt-ft*. At page $2-7$, we observe the DORS histograms consistently "flatter" than DNN-*rt-ft* histograms. Note that from page $2-7$ DORS wins most cases against DNN-*rt-ft*. This could be explained by the fact that the *revised*-Thompson is better in capturing user online intents so the online ranking results are optimized and people tend to click more frequently. Finally, the DORS histogram at page-8 resembles DNN-*rt-ft* again, due to the fact that at page-8 most users either have their intents captured or they abandon the site visit.

4.5 System Specifications

Our current DORS ranking framework is maintained by hundreds of Linux servers[6]. Each machine processes 192 queries per second in average (peaking at 311), and the 99% percentile end-to-end latency is within 200.0 milliseconds. In our current production, the feature dimension is 5.369×10^8.

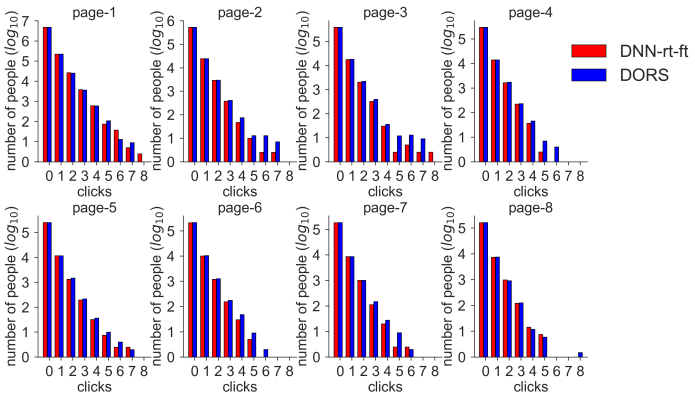

Fig. 7. Daily page-wise click distribution: DORS v.s. DNN-*rt-ft*.

[6] We could not release the exact number of operating servers due to the company confidentiality.

5 Conclusion

In this paper, we presented a novel three-level recommender system for e-commerce recommendation, which includes candidate retrieval, learning-to-rank DNN and MAB based online re-ranking. Compared to the traditional recommender systems, our approach is relevant, responsive, scalable and is running in production to serve millions of recommendations everyday. Our offline case studies have empirically demonstrated the efficiency for learning-to-rank DNN serving as the static ranking as well as the warm start for the revised-Thompson initializations that enable the quick convergence. Furthermore, online A/B experiments have been used to prove DORS is superior in terms of page-wise / overall NDCG as well as the operating revenue gains when serving in the real production system.

References

1. Agarwal, D., Chen, B.C., Elango, P.: Fast online learning through offline initialization for time-sensitive recommendation. In: KDD. pp. 703–712. ACM (2010)
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine learning **47**(2-3), 235–256 (2002)
3. Basak, D., Pal, S., Patranabis, D.C.: Support vector regression. Neural Information Processing-Letters and Reviews **11**(10), 203–224 (2007)
4. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: ICML. pp. 89–96. ACM (2005)
5. Cao, Z., Qin, T., Liu, T.Y., Tsai, M.F., Li, H.: Learning to rank: from pairwise approach to listwise approach. In: ICML. pp. 129–136. ACM (2007)
6. Chang, S., Zhang, Y., Tang, J., Yin, D., Chang, Y., Hasegawa-Johnson, M.A., Huang, T.S.: Streaming recommender systems. In: WWW. pp. 381–389 (2017)
7. Chen, C., Yin, H., Yao, J., Cui, B.: Terec: A temporal recommender system over tweet stream. VLDB **6**(12), 1254–1257 (2013)
8. Cherkassky, V., Ma, Y.: Practical selection of svm parameters and noise estimation for svm regression. Neural Networks **17**(1), 113–126 (2004)
9. Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., Gupta, S., He, Y., Lambert, M., Livingston, B., et al.: The youtube video recommendation system. In: Recsys. pp. 293–296. ACM (2010)
10. Diaz-Aviles, E., Drumond, L., Schmidt-Thieme, L., Nejdl, W.: Real-time top-n recommendation in social streams. In: Recsys. pp. 59–66. ACM (2012)
11. Ding, Y., Li, X.: Time weight collaborative filtering. In: CIKM. pp. 485–492. ACM (2005)
12. Gao, H., Tang, J., Hu, X., Liu, H.: Exploring temporal effects for location recommendation on location-based social networks. In: Recsys. pp. 93–100. ACM (2013)
13. Gomez-Uribe, C.A., Hunt, N.: The netflix recommender system: Algorithms, business value, and innovation. TMIS **6**(4), 13 (2016)
14. Guan, Z., Bu, J., Mei, Q., Chen, C., Wang, C.: Personalized tag recommendation using graph-based ranking on multi-type interrelated objects. In: SIGIR. pp. 540–547. ACM (2009)
15. Gultekin, S., Paisley, J.: A collaborative kalman filter for time-evolving dyadic processes. In: ICDM. pp. 140–149. IEEE (2014)

16. Hannon, J., Bennett, M., Smyth, B.: Recommending twitter users to follow using content and collaborative filtering approaches. In: Recsys. pp. 199–206. ACM (2010)
17. Koren, Y.: Collaborative filtering with temporal dynamics. Communications of the ACM **53**(4), 89–97 (2010)
18. Lang, K.: Newsweeder: Learning to filter netnews. In: in Proceedings of the 12th International Machine Learning Conference (ML95 (1995)
19. Langford, J., Zhang, T.: The epoch-greedy algorithm for multi-armed bandits with side information. In: NIPS. pp. 817–824 (2008)
20. Liu, Y., Miao, J., Zhang, M., Ma, S., Ru, L.: How do users describe their information need: Query recommendation based on snippet click model. Expert Systems with Applications **38**(11), 13847 – 13856 (2011)
21. Lu, D., Voss, C., Tao, F., Ren, X., Guan, R., Korolov, R., Zhang, T., Wang, D., Li, H., Cassidy, T., et al.: Cross-media event extraction and recommendation. In: NAACL. pp. 72–76 (2016)
22. Lu, Z., Agarwal, D., Dhillon, I.S.: A spatio-temporal approach to collaborative filtering. In: Recsys. pp. 13–20. ACM (2009)
23. Rendle, S.: Factorization machines. In: ICDM. pp. 995–1000. IEEE (2010)
24. Rendle, S., Freudenthaler, C., Schmidt-Thieme, L.: Factorizing personalized markov chains for next-basket recommendation. In: WWW. pp. 811–820. ACM (2010)
25. Rendle, S., Schmidt-Thieme, L.: Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In: Recsys. pp. 251–258. ACM (2008)
26. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: WWW. pp. 285–295. ACM (2001)
27. Severyn, A., Moschitti, A.: Learning to rank short text pairs with convolutional deep neural networks. In: SIGIR. pp. 373–382. ACM (2015)
28. Tang, J., Hu, X., Gao, H., Liu, H.: Exploiting local and global social context for recommendation. In: IJCAI. vol. 13, pp. 2712–2718 (2013)
29. Tang, J., Hu, X., Liu, H.: Social recommendation: a review. Social Network Analysis and Mining **3**(4), 1113–1133 (2013)
30. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. Biometrika **25**(3/4), 285–294 (1933)
31. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, University of Cambridge England (1989)
32. Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., Sun, J.: Temporal recommendation on graphs via long-and short-term preference fusion. In: KDD. pp. 723–732. ACM (2010)
33. Xiong, L., Chen, X., Huang, T.K., Schneider, J., Carbonell, J.G.: Temporal collaborative filtering with bayesian probabilistic tensor factorization. In: SDM. pp. 211–222. SIAM (2010)
34. Yin, D., Hong, L., Xue, Z., Davison, B.D.: Temporal dynamics of user interests in tagging systems. In: AAAI (2011)
35. Yu, X., Ren, X., Sun, Y., Gu, Q., Sturt, B., Khandelwal, U., Norick, B., Han, J.: Personalized entity recommendation: A heterogeneous information network approach. In: WSDM. pp. 283–292. ACM (2014)
36. Zhang, Y., Zhang, M., Liu, Y., Ma, S., Feng, S.: Localized matrix factorization for recommendation based on matrix block diagonal forms. In: WWW (2013)