Top-K Off-Policy Correction for a REINFORCE Recommender System

Minmin Chen*, Alex Beutel*, Paul Covington*, Sagar Jain, Francois Belletti, Ed H. Chi
Google, Inc.
Mountain View, CA
minminc,alexbeutel,pcovington,sagarj,belletti,edchi@google.com

ABSTRACT

Industrial recommender systems deal with *extremely large* action spaces – many millions of items to recommend. Moreover, they need to serve billions of users, who are unique at any point in time, making a complex user state space. Luckily, huge quantities of logged implicit feedback (*e.g.*, user clicks, dwell time) are available for learning. Learning from the logged feedback is however subject to biases caused by only observing feedback on recommendations selected by the previous versions of the recommender. In this work, we present a general recipe of addressing such biases in a production top-K recommender system at YouTube, built with a policy-gradient-based algorithm, i.e. REINFORCE [48]. The contributions of the paper are: (1) scaling REINFORCE to a production recommender system with an action space on the orders of millions; (2) applying off-policy correction to address data biases in learning from logged feedback collected from multiple behavior policies; (3) proposing a novel top-K off-policy correction to account for our policy recommending multiple items at a time; (4) showcasing the value of exploration. We demonstrate the efficacy of our approaches through a series of simulations and multiple live experiments on YouTube.

ACM Reference Format:
Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, Ed H. Chi. 2019. Top-K Off-Policy Correction for a REINFORCE Recommender System. In *The Twelfth ACM International Conference on Web Search and Data Mining (WSDM' 19), February 11-15, 2019, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3289600.3290999

1 INTRODUCTION

Recommender systems are relied on, throughout industry, to help users sort through huge corpuses of content and discover the small fraction of content they would be interested in. This problem is challenging because of the huge number of items that could be recommended. Furthermore, surfacing the right item to the right user at the right time requires the recommender system to constantly adapt to users' shifting interest (state) based on their historical interaction with the system [6]. Unfortunately, we observe relatively little data for such a large state and action space, with most users only having been exposed to a small fraction of items and providing explicit feedback to an even smaller fraction. That is, recommender systems receive extremely sparse data for training in general, e.g., the Netflix Prize dataset was only 0.1% dense [5]. As a result, a good amount of research in recommender systems explores different mechanisms for treating this extreme sparsity. Learning from implicit user feedback, such as clicks and dwell-time, as well as filling in unobserved interactions, has been an important step in improving recommenders [19] but the problem remains an open one.

In a mostly separate line of research, reinforcement learning (RL) has recently achieved impressive advances in games [38, 46] as well as robotics [22, 25]. RL in general focuses on building agents that take actions in an environment so as to maximize some notion of long term reward. Here we explore framing recommendation as building RL agents to maximize each user's long term satisfaction with the system. This offers us new perspectives on recommendation problems as well as opportunities to build on top of the recent RL advancement. However, there are significant challenges to put this perspective into practice.

As introduced above, recommender systems deal with large state and action spaces, and this is particularly exacerbated in industrial settings. The set of items available to recommend is non-stationary and new items are brought into the system constantly, resulting in an ever-growing action space with new items having even sparser feedback. Further, user preferences over these items are shifting all the time, resulting in continuously-evolving user states. Being able to reason through these large number of actions in such a complex environment poses unique challenges in applying existing RL algorithms. Here we share our experience adapting the REINFORCE algorithm [48] to a neural candidate generator (a top-K recommender system) with extremely large action and state spaces.

In addition to the massive action and state spaces, RL for recommendation is distinct in its limited availability of data. Classic RL applications have overcome data inefficiencies by collecting large quantities of training data with self-play and simulation [38]. In contrast, the complex dynamics of the recommender system has made simulation for generating realistic recommendation data nonviable. As a result, we cannot easily probe for reward in previously unexplored areas of the state and action space, since observing reward requires giving a real recommendation to a real user. Instead, the model relies mostly on data made available from the previous recommendation models (policies), most of which we cannot control or can no longer control. To most effectively utilize logged-feedback from other policies, we take an off-policy learning

* Authors contributed equally.

WSDM '19, February 11–15, 2019, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-5940-5/19/02.
https://doi.org/10.1145/3289600.3290999

approach, in which we simultaneously learn a model of the previous policies and incorporate it in correcting the data biases when training our new policy. We also experimentally demonstrate the value in exploratory data.

Finally, most of the research in RL focuses on producing a policy that chooses a single item. Real-world recommenders, on the other hand, typically offer the user multiple recommendations at a time [44]. Therefore, we define a novel top-K off-policy correction for our top-K recommender system. We find that while the standard off-policy correction results in a policy that is optimal for top-1 recommendation, this top-K off-policy correction leads to significant better top-K recommendations in both simulations and live experiments. Together, we offer the following contributions:

- **REINFORCE Recommender:** We scale a REINFORCE policy-gradient-based approach to learn a neural recommendation policy in a extremely large action space.
- **Off-Policy Candidate Generation:** We apply off-policy correction to learn from logged feedback, collected from an ensemble of prior model policies. We incorporate a learned neural model of the behavior policies to correct data biases.
- **Top-K Off-Policy Correction:** We offer a novel top-K off-policy correction to account for the fact that our recommender outputs multiple items at a time.
- **Benefits in Live Experiments:** We demonstrate in live experiments, which was rarely done in existing RL literature, the value of these approaches to improve user long term satisfaction.

We find this combination of approaches valuable for increasing user enjoyment and believe it frames many of the practical challenges going forward for using RL in recommendations.

2 RELATED WORK

Reinforcement Learning: Value-based approaches such as Q-learning, and policy-based ones such as policy gradients constitute classical approaches to solve RL problems [40]. A general comparison of modern RL approaches can be found in [29] with a focus on asynchronous learning which is key to scaling up to large problems. Although value-based methods present many advantages such as seamless off-policy learning, they are known to be prone to instability with function approximation [41]. Often, extensive hyper-parameter tuning is required to achieve stable behavior for these approaches. Despite the practical success of many value-based approaches such as deep Q-learning [30], policy convergence of these algorithms are not well-studied. Policy-based approaches on the other hand, remain rather stable w.r.t. function approximations given a sufficiently small learning rate. We therefore choose to rely on a policy-gradient-based approach, in particular REINFORCE [48], and to adapt this on-policy method to provide reliable policy gradient estimates when training off-policy.

Neural Recommenders: Another line of work that is closely related to ours is the growing body of literature on applying deep neural networks to recommender systems [11, 16, 37], in particular using recurrent neural networks to incorporate temporal information and historical events for recommendation [6, 17, 20, 45, 49]. We employed similar network architectures to model the evolving of user states through interactions with the recommender system. As neural architecture design is not the main focus of our work, we refer interested readers to these prior works for more detailed discussions.

Bandit Problems in recommender systems: On-line learning methods are also popular to quickly adapt recommendation systems as new user feedback becomes available. Bandit algorithms such as Upper Confidence Bound (UCB) [3] trade off exploration and exploitation in an analytically tractable way that provides strong guarantees on the regret. Different algorithms such as Thomson sampling [9], have been successfully applied to news recommendations and display advertising. Contextual bandits offer a context-aware refinement of the basic on-line learning approaches and tailor the recommendation toward user interests [27]. Agarwal et al. [2] aimed to make contextual bandits tractable and easy to implement. Hybrid methods that rely on matrix factorization and bandits have also been developed to solve cold-start problems in recommender systems [28].

Propensity Scoring and Reinforcement Learning in Recommender Systems: The problem of learning off-policy [31, 33, 34] is pervasive in RL and affects policy gradient generally. As a policy evolves so does the distribution under which gradient expectations are computed. Standard approaches in robotics [1, 36] circumvent this issue by constraining policy updates so that they do not change the policy too substantially before new data is collected under an updated policy, which in return provides monotonic improvement guarantees of the RL objective. Such proximal methods are unfortunately not applicable in the recommendations setting where item catalogues and user behaviors change rapidly, and therefore substantial policy changes are required. Meanwhile feedback is slow to collect at scale w.r.t. the large state and action space. As a matter of fact, offline evaluation of a given policy is already a challenge in the recommender system setting. Multiple off-policy estimators leveraging inverse-propensity scores, capped inverse-propensity scores and various variance control measures have been developed [13, 42, 43, 47]. Off-policy evaluation corrects for a similar data skew as off-policy RL and similar methods are applied on both problems. Inverse propensity scoring has also been employed to improve a serving policy at scale in [39]. Joachims et al. [21] learns a model of logged feedback for an unbiased ranking model; we take a similar perspective but use a DNN to model the logged behavior policy required for the off-policy learning. More recently an off-policy approach has been adapted to the more complex problem of slate recommendation [44] where a pseudo-inverse estimator assuming a structural prior on the slate and reward is applied in conjunction with inverse propensity scoring.

3 REINFORCE RECOMMENDER

We begin with describing the setup of our recommender system, and our approach to RL-based recommendation.

For each user, we consider a sequence of user historical interactions with the system, recording the actions taken by the recommender, *i.e.,* videos recommended, as well as user feedback, such as clicks and watch time. Given such a sequence, we predict the next action to take, *i.e.*, videos to recommend, so that user satisfaction metrics, *e.g.*, indicated by clicks or watch time, improve.

We translate this setup into a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \gamma)$ where

- $\mathcal{S}$: a continuous state space describing the user states;
- $\mathcal{A}$: a discrete action space, containing items available for recommendation;
- $\mathbf{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the state transition probability;
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $r(s, a)$ is the immediate reward obtained by performing action a at user state s;
- ρ_0 is the initial state distribution;
- γ is the discount factor for future rewards.

We seek a policy $\pi(a|s)$ that casts a distribution over the item to recommend $a \in \mathcal{A}$ conditional to the user state $s \in \mathcal{S}$, so as to maximize the expected cumulative reward obtained by the recommender system,

$$\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right], \text{ where } R(\tau) = \sum_{t=0}^{|\tau|} r(s_t, a_t)$$

Here the expectation is taken over the trajectories $\tau = (s_0, a_0, s_1, \cdots)$ obtained by acting according to the policy: $s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathbf{P}(\cdot|s_t, a_t)$.

Different families of methods are available to solve such an RL problems: Q-learning [38], Policy Gradient [26, 36, 48] and black box optimization [15]. Here we focus on a policy-gradient-based approach, *i.e.,* REINFORCE [48].

We assume a function form of the policy π_θ, parametrised by $\theta \in \mathbb{R}^d$. The gradient of the expected cumulative reward with respect to the policy parameters can be derived analytically thanks to the "log-trick", yielding the following REINFORCE gradient

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right] \tag{1}$$

In on-line RL, where the policy gradient is computed on trajectories generated by the policy under consideration, the estimator of the policy gradient is unbiased and decomposable as:

$$\sum_{\tau\sim\pi_\theta} R(\tau)\nabla_\theta \log \pi_\theta(\tau) \approx \sum_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{|\tau|} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{2}$$

The approximation results from replacing $R(\tau)$ with a discounted future reward $R_t = \sum_{t'=t}^{|\tau|} \gamma^{t'-t} r(s_{t'}, a_{t'})$ for action at time t to reduce variance in the gradient estimate.

4 OFF-POLICY CORRECTION

Unlike classical reinforcement learning, our learner does not have real-time interactive control of the recommender due to learning and infrastructure constraints. In other words, we cannot perform online updates to the policy and generate trajectories according to the updated policy immediately. Instead we receive logged feedback of actions chosen by a historical policy (or a mixture of policies), which could have a different distribution over the action space than the policy we are updating.

We focus on addressing the data biases that arise when applying policy gradient methods under this setting. In particular, the fact that we collect data with a periodicity of several hours and compute many policy parameter updates before deploying a new version of the policy in production implies that the set of trajectories we employ to estimate the policy gradient is generated by a different policy. Moreover, we learn from batched feedback collected by other recommenders as well, which follow drastically different policies. A naive policy gradient estimator is no longer unbiased as the gradient in Equation (2) requires sampling trajectories from the updated policy π_θ while the trajectories we collected were drawn from a combination of historical policies β.

We address the distribution mismatch with importance weighting [31, 33, 34]. Consider a trajectory $\tau = (s_0, a_0, s_1, \cdots)$ sampled according to a behavior policy β, the off-policy-corrected gradient estimator is then:

$$\sum_{\tau\sim\beta} \frac{\pi_\theta(\tau)}{\beta(\tau)}\left[\sum_{t=0}^{|\tau|} R_t \nabla_\theta \log\left(\pi_\theta(a_t|s_t)\right)\right].$$

where

$$\frac{\pi_\theta(\tau)}{\beta(\tau)} = \frac{\rho(s_0)\prod_{t=0}^{|\tau|}\mathbf{P}(s_{t+1}|s_t, a_t)\pi(a_t|s_t)}{\rho(s_0)\prod_{t=0}^{|\tau|}\mathbf{P}(s_{t+1}|s_t, a_t)\beta(a_t|s_t)} = \prod_{t=0}^{|\tau|}\frac{\pi(a_t|s_t)}{\beta(a_t|s_t)}.$$

is the importance weight. This correction produces an unbiased estimator whenever the trajectories are collected with actions sampled according to β. However, the variance of the estimator is huge due to the chained products, leading quickly to very low or high values of the importance weights.

To reduce the variance of each gradient term corresponding to a time t on the trajectory, we approximate the importance weights by first ignoring the terms after time t in the chained products, and further taking a first-order approximation:

$$\prod_{t'=0}^{|\tau|}\frac{\pi(a_{t'}|s_{t'})}{\beta(a_{t'}|s_{t'})} \approx \prod_{t'=0}^{t}\frac{\pi(a_{t'}|s_{t'})}{\beta(a_{t'}|s_{t'})} = \frac{P_{\pi_\theta}(s_t)}{P_\beta(s_t)}\frac{\pi(a_t|s_t)}{\beta(a_t|s_t)} \approx \frac{\pi(a_t|s_t)}{\beta(a_t|s_t)}.$$

which yields a biased estimator of the policy gradient with lower variance:

$$\sum_{\tau\sim\beta}\left[\sum_{t=0}^{|\tau|}\frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]. \tag{3}$$

Achiam et al. [1] prove that the impact of this first-order approximation on the total reward of the learned policy is bounded in magnitude by $O\left(E_{s\sim d^\beta}\left[D_{TV}(\pi|\beta)[s]\right]\right)$ where D_{TV} is the total variation between $\pi(\cdot|s)$ and $\beta(\cdot|s)$ and d^β is the discounted future state distribution under β. This estimator trades off the variance of the exact off-policy correction while still correcting for the large bias of a non-corrected policy gradient, which is better suited for on-policy learning.

4.1 Parametrising the policy π_θ

We model our belief on the user state at each time t, which capture both evolving user interests using a n-dimensional vector, that is, $\mathbf{s}_t \in \mathbb{R}^n$. The action taken at each time t along the trajectory is embedded using an m-dimensional vector $\mathbf{u}_{a_t} \in \mathbb{R}^m$. We model the state transition $\mathbf{P} : \mathcal{S}\times\mathcal{A}\times\mathcal{S}$ with a recurrent neural network [6, 49]:

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{u}_{a_t}).$$

We experimented with a variety of popular RNN cells such as Long Short-Term Memory (LSTM) [18] and Gated Recurrent Units (GRU) [10], and ended up using a simplified cell called Chaos Free

RNN (CFN) [24] due to its stability and computational efficiency. The state is updated recursively as

$$\begin{aligned} \mathbf{s}_{t+1} &= \mathbf{z}_t \odot \tanh(\mathbf{s}_t) + \mathbf{i}_t \odot \tanh(\mathbf{W}_a \mathbf{u}_{a_t}) \\ \mathbf{z}_t &= \sigma(\mathbf{U}_z \mathbf{s}_t + \mathbf{W}_z \mathbf{u}_{a_t} + \mathbf{b}_z) \\ \mathbf{i}_t &= \sigma(\mathbf{U}_i \mathbf{s}_t + \mathbf{W}_i \mathbf{u}_{a_t} + \mathbf{b}_i) \end{aligned} \tag{4}$$

where $\mathbf{z}_t, \mathbf{i}_t \in \mathbb{R}^n$ are the update and input gate respectively.

Conditioning on a user state $\mathbf{s}$, the policy $\pi_\theta(a|\mathbf{s})$ is then modeled with a simple softmax,

$$\pi_\theta(a|\mathbf{s}) = \frac{exp(\mathbf{s}^\top \mathbf{v}_a / T)}{\sum_{a' \in \mathcal{A}} exp(\mathbf{s}^\top \mathbf{v}_{a'} / T)} \tag{5}$$

where $\mathbf{v}_a \in \mathbb{R}^n$ is another embedding for each action a in the action space $\mathcal{A}$ and T is a temperature that is normally set to 1. Using a higher value in T produces a smoother policy over the action space. The normalization term in the softmax requires going over all the possible actions, which is in the order of millions in our setting. To speed up the computation, we perform sampled softmax [4] during training. At serving time, we used an efficient nearest neighbor search algorithm to retrieve top actions and approximate the softmax probability using these actions only, as detailed in section 5.

In summary, the parameter θ of the policy π_θ contains the two action embeddings $\mathbf{U} \in \mathbb{R}^{m \times |\mathcal{A}|}$ and $\mathbf{V} \in \mathbb{R}^{n \times |\mathcal{A}|}$ as well as the weight matrices $\mathbf{U}_z, \mathbf{U}_i \in \mathbb{R}^{n \times n}$, $\mathbf{W}_u, \mathbf{W}_i, \mathbf{W}_a \in \mathbb{R}^{n \times m}$ and biases $\mathbf{b}_u, \mathbf{b}_i \in \mathbb{R}^n$ in the RNN cell. Figure 1 shows a diagram describing the neural architecture of the main policy π_θ. Given an observed trajectory $\tau = (s_0, a_0, s_1, \cdots)$ sampled from a behavior policy β, the new policy first generates a model of the user state $\mathbf{s}_{t+1}$ by starting with an initial state $\mathbf{s}_0 \sim \rho_0$[1] and iterating through the recurrent cell as in Equation (4)[2]. Given the user state $\mathbf{s}_{t+1}$ the policy head casts a distribution on the action space through a softmax as in Equation (5). With $\pi_\theta(a_{t+1}|\mathbf{s}_{t+1})$, we can then produce a policy gradient as in Equation (3) to update the policy.

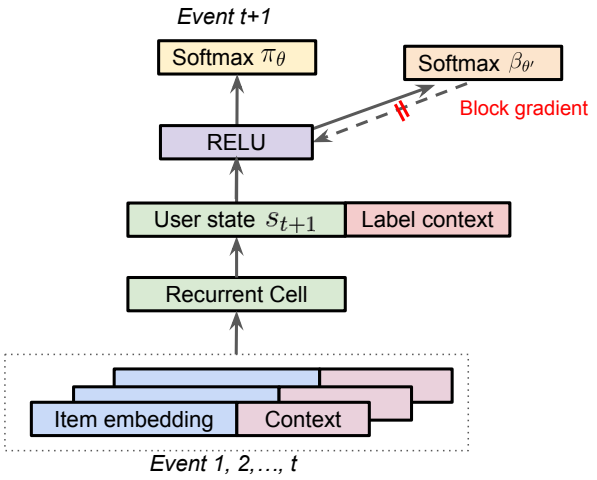

Figure 1: A diagram shows the parametrisation of the policy π_θ as well as the behavior policy $\beta_{\theta'}$.

[1]In our experiment, we used a fixed initial state distribution, where $\mathbf{s}_0 = \mathbf{0} \in \mathbb{R}^n$

[2]We take into account of the context [6] of the action, such as page, device and time information, as input to the RNN cell besides the action embedding itself.

4.2 Estimating the behavior policy β

One difficulty in coming up with the off-policy corrected estimator in Equation (3) is to get the behavior policy β. Ideally, for each logged feedback of a chosen action we received, we would like to also log the probability of the behavior policy choosing that action. Directly logging the behavior policy is however not feasible in our case as (1) there are multiple agents in our system, many of which we do not have control over, and (2) some agents have a deterministic policy, and setting β to 0 or 1 is not the most effective way to utilize these logged feedback.

Instead we take the approach first introduced in [39], and estimate the behavior policy β, which in our case is a mixture of the policies of the multiple agents in the system, using the logged actions. Given a set of logged feedback $\mathcal{D} = \{(\mathbf{s}_i, a_i), i = 1, \cdots, N\}$, Strehl et al. [39] estimates $\hat{\beta}(a)$ independent of user state by aggregate action frequency throughout the corpus. In contrast, we adopt a context-dependent neural estimator. For each state-action pair (s, a) collected, we estimate the probability $\hat{\beta}_{\theta'}(a|s)$ that the mixture of behavior policies choosing that action using another softmax, parametrised by θ'. As shown in Figure 1, we re-use the user state s generated from the RNN model from the main policy, and model the mixed behavior policy with another softmax layer. To prevent the behavior head from intefering with the user state of the main policy, we block its gradient from flowing back into the RNN. We also experimented with separating the π_θ and $\beta_{\theta'}$ estimators, which incurs computational overhead for computing another state representation but does not results in any metric improvement in offline and live experiments.

Despite a substantial sharing of parameters between the two policy heads π_θ and $\beta_{\theta'}$, there are two noticeable difference between them: (1) While the main policy π_θ is effectively trained using a weighted softmax to take into account of long term reward, the behavior policy head $\beta_{\theta'}$ is trained using only the state-action pairs; (2) While the main policy head π_θ is trained using only items on the trajectory with non-zero reward [3], the behavior policy $\beta_{\theta'}$ is trained using all of the items on the trajectory to avoid introducing bias in the β estimate.

In [39], it is argued that that a behavior policy that is deterministically choosing an action a given state s at time t_1 and action b at time t_2 can be treated as randomizing between action a and b over the timespan of the logging. Here we could argue the same point, which explains why the behavior policy could be other than 0 or 1 given a deterministic policy. In addition, since we have multiple policies acting simultaneously, if one policy is determinstically choosing action a given user state s, and another one is determinstically choosing action b, then estimating $\hat{\beta}_{\theta'}$ in such a way would approximate the expected frequency of action a being chosen under the mixture of these behavior policies given user state s.

4.3 Top-K Off-Policy Correction

Another challenge in our setting is that our system recommends a page of k items to users at a time. As users are going to browse

[3]1. Actions with zero-reward will not contribute to the gradient update in π_θ; 2. We ignore them in the user state update as users are unlikely to notice them and as a result, we assume the user state are not influenced by these actions; 3. It saves computational cost.

through (the full or partial set of) our recommendations and potentially interact with more than one item, we need to pick a set of relevant items instead of a single one. In other words, we seek a policy $\Pi_\Theta(A|s)$, here each action A is to select a set of k items, to maximize the expected cumulative reward,

$$\max_\Theta \mathbb{E}_{\tau\sim\Pi_\theta}\left[\sum_t r(s_t, A_t)\right].$$

The trajectory $\tau = (s_0, A_0, s_1, \cdots)$ is obtained through acting according to $s_0 \sim \rho_0, A_t \sim \Pi(|s_t), s_{t+1} \sim \mathbf{P}(\cdot|s_t, A_t)$. Unfortunately, the action space grows exponentially under this set recommendation formulation [44, 50], which is prohibitively large given the number of items we choose from are in the orders of millions.

To make the problem tractable, we assume that the expected reward of a set of **non-repetitive** items equal to the sum of the expected reward of each item in the set [4]. Furthermore, we constrain ourselves to generate the set action A by independently sampling each item a according to the softmax policy π_θ described in Equation (5) and then de-duplicate. That is,

$$\Pi_\Theta(A'|s) = \prod_{a\in A'} \pi_\theta(a|s).$$

Note that set A' will contain duplicate items, which are removed to form a non-repetitive set A.

Under these assumptions, we can adapt the REINFORCE algorithm to the set recommendation setting by simply modifying the gradient update in Equation (2) to

$$\sum_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{|\tau|} R_t \nabla_\theta \log \alpha_\theta(a_t|s_t)\right]$$

where $\alpha_\theta(a|s) = 1 - (1-\pi_\theta(a|s))^K$ is the probability that an item a appears in the final non-repetitive set A. Here $K = |A'| > |A| = k$ [5].

We can then update the off-policy corrected gradient in Equation (3) by replacing π_θ with α_θ, resulting in the top-K off-policy correction factor:

$$\begin{aligned}&\sum_{\tau\sim\beta}\left[\sum_{t=0}^{|\tau|} \frac{\alpha_\theta(a_t|s_t)}{\beta(a_t|s_t)} R_t \nabla_\theta \log \alpha_\theta(a_t|s_t)\right] \\ =&\sum_{\tau\sim\beta}\left[\sum_{t=0}^{|\tau|} \frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)} \frac{\partial\alpha(a_t|s_t)}{\partial\pi(a_t|s_t)} R_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right].\end{aligned} \tag{6}$$

Comparing Equation (6) with Equation (3), the top-K policy adds an additional multiplier of

$$\lambda_K(s_t, a_t) = \frac{\partial\alpha(a_t|s_t)}{\partial\pi(a_t|s_t)} = K(1-\pi_\theta(a_t|s_t))^{K-1} \tag{7}$$

to the original off-policy correction factor of $\frac{\pi(a|s)}{\beta(a|s)}$.

Now let us take a closer look at this additional multiplier:

- As $\pi_\theta(a|s) \to 0$, $\lambda_K(s,a) \to K$. The top-K off-policy correction increases the policy update by a factor of K comparing to the standard off-policy correction;
- As $\pi_\theta(a|s) \to 1$, $\lambda_K(s,a) \to 0$. This multiplier zeros out the policy update.
- As K increases, this multiplier reduces the gradient to zero faster as $\pi_\theta(a|s)$ reaches a reasonable range.

In summary, when the desirable item has a small mass in the softmax policy $\pi_\theta(\cdot|s)$, the top-K correction more aggressively pushes up its likelihood than the standard correction. Once the softmax policy $\pi_\theta(\cdot|s)$ casts a reasonable mass on the desirable item (to ensure it will be likely to appear in the top-K), the correction then zeros out the gradient and no longer tries to push up its likelihood. This in return allows other items of interest to take up some mass in the softmax policy. As we are going to demonstrate in the simulation as well as live experiment, while the standard off-policy correction converges to a policy that is optimal when choosing a single item, the top-K correction leads to better top-K recommendations.

4.4 Variance Reduction Techniques

As detailed at the beginning of this section, we take a first-order approximation to reduce variance in the gradient estimate. Nonetheless, the gradient can still suffer from large variance due to large importance weight of $\omega(s,a) = \frac{\pi(a|s)}{\beta(a|s)}$ as shown in Equation (3), Similarly for top-K off-policy correction. Large importance weight could result from (1) large deviation of the new policy $\pi(\cdot|s)$ from the behavior policy, in particular, the new policy explores regions that are less explored by the behavior policy. That is, $\pi(a|s) \gg \beta(a|s)$ and (2) large variance in the β estimate.

We tested several techniques proposed in counterfactual learning and RL literature to control variance in the gradient estimate. Most of these techniques reduce variance at the cost of introducing some bias in the gradient estimate.

Weight Capping. The first approach we take is to simply cap the weight [8] as

$$\bar{\omega}_c(s,a) = \min\left(\frac{\pi(a|s)}{\beta(a|s)}, c\right). \tag{8}$$

Smaller value of c reduces variance in the gradient estimate, but introduces larger bias.

Normalized Importance Sampling (NIS). Second technique we employed is to introduce a ratio control variate, where we use classical weight normalization [32] defined by:

$$\bar{\omega}_n(s,a) = \frac{\omega(s,a)}{\sum_{(s',a')\sim\beta}\omega(s',a')}.$$

As $\mathbb{E}_\beta[\omega(s,a)] = 1$, the normalizing constant is equal to n, the batch size, in expectation. As n increases, the effect of NIS is equivalent to tuning down the learning rate.

Trusted Region Policy Optimization (TRPO). TRPO [36] prevents the new policy π from deviating from the behavior policy by adding a regularization that penalizes the KL divergence of these two policies. It achieves similar effect as the weight capping.

5 EXPLORATION

As should be clear by this point, the distribution of training data is important for learning a good policy. Exploration policies to inquire about actions rarely taken by the existing system have been extensively studied. In practice, brute-force exploration, such as ϵ-greedy, is not viable in a production system like YouTube

[4] This assumption holds if users inspect each item on a page independently.
[5] At a result of the sampling with replacement and de-duplicate, the size k of the final set A can vary.

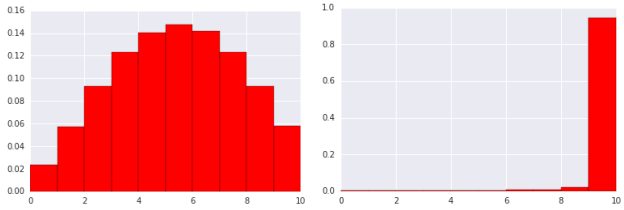

Figure 2: learned policy π_θ when behavior policy β is skewed to favor the actions with least reward, *i.e.*, $\beta(a_i) = \frac{11-i}{55}, \forall i = 1, \cdots, 10$. (left): without off-policy correction; (right): with off-policy correction.

where this could, and mostly likely would, result in inappropriate recommendations and a bad user experience. For example, Schnabel et al. [35] studied the cost of exploration.

Instead we employ Boltzmann exploration [12] to get the benefit of exploratory data without negatively impacting user experience. We consider using a stochastic policy where recommendations are sampled from π_θ rather than taking the K items with the highest probability. This has the challenge of being computationally inefficient because we need to calculate the full softmax, which is prohibitively expensive considering our action space. Rather, we make use of efficient approximate nearest neighbor-based systems to look up the top M items in the softmax [14]. We then feed the logits of these M items into a smaller softmax to normalize the probabilities and sample from this distribution. By setting $M \gg K$ we can still retrieve most of the probability mass, limit the risk of bad recommendations, and enable computationally efficient sampling. In practice, we further balance exploration and exploitation by returning the top K' most probable items and sample $K - K'$ items from the remaining $M - K'$ items.

6 EXPERIMENTAL RESULTS

We showcase the effectiveness of these approaches for addressing data biases in a series of simulated experiments and live experiments in an industrial-scale recommender system.

6.1 Simulation

We start with designing simulation experiments to shed light on the off-policy correction ideas under more controlled settings. To simplify our simulation, we assume the problem is stateless, in other words, the reward R is independent of user states, and the action does not alter the user states either. As a result, each action on a trajectory can be independently chosen.

6.1.1 Off-policy correction. In the first simulation, we assume there are 10 items, that is $\mathcal{A} = \{a_i, i = 1, \cdots, 10\}$. The reward of each one is equal to its index, that is, $r(a_i) = i$. When we are choosing a single item, the optimal policy under this setting is to always choose the 10^{th} item as it gives the most reward, that is,

$$\pi^*(a_i) = \mathbb{I}(i = 10).$$

We parameterize π_θ using a stateless softmax

$$\pi(a_i) = \frac{e^{\theta_i}}{\sum_j e^{\theta_j}}$$

Given observations sampled from the behavior policy β, naively applying policy gradient without taking into account of data bias as in Equation (1) would converge to a policy

$$\pi(a_i) = \frac{r(a_i)\beta(a_i)}{\sum_j r(a_j)\beta(a_j)}$$

This has an obvious downside: the more the behavior policy chooses a sub-optimal item, the more the new policy will be biased toward choosing the same item.

Figure 2 compares the policies π_θ, learned without and with off-policy correction using SGD [7], when the behavior policy β is skewed to favor items with least reward. As shown in Figure 2 (left), naively applying the policy gradient without accounting for the data biases leads to a sub-optimal policy. In the worst case, if the behavior policy always chooses the action with the lowest reward, we will end up with a policy that is arbitrarily poor and mimicking the behavior policy (*i.e.*, converge to selecting the least rewarded item). On the other hand, applying the off-policy correction allows us to converge to the optimal policy π^* regardless of how the data is collected, as shown in Figure 2 (right).

6.1.2 Top-K off-policy correction. To understand the difference between the standard off-policy correction and the top-K off-policy correction proposed, we designed another simulation in which we can recommend multiple items. Again we assume there are 10 items, with $r(a_1) = 10, r(a_2) = 9$, and the remaining items are of much lower reward $r(a_i) = 1, \forall i = 3, \cdots, 10$. Here we focus on recommending two items, that is, $K = 2$. The behavior policy β follows a uniform distribution, *i.e.*, choosing each item with equal chance.

Given an observation (a_i, r_i) sampled from β, the standard off-policy correction has a SGD updates of the following form,

$$\theta_j = \theta_j + \eta \frac{\pi_\theta(a_j)}{\beta(a_j)} r(a_i) \left[\mathbb{I}(j = i) - \pi_\theta(a_j)\right], \quad \forall j = 1, \cdots, 10$$

where η is the learning rate. SGD keeps increasing the likelihood of the item a_i proportional to the expected reward under π_θ until $\pi_\theta(a_i) = 1$, under which the gradient goes to 0. The top-K off-policy correction, on the other hand, has an update of the following form,

$$\theta_j = \theta_j + \eta \lambda_K(a_i) \frac{\pi_\theta(a_j)}{\beta(a_j)} r(a_i) \left[\mathbb{I}(j = i) - \pi_\theta(a_j)\right], \quad \forall j = 1, \cdots, 10$$

where $\lambda_K(a_i)$ is the multiplier as defined in section 4.3. When $\pi_\theta(a_i)$ is small, $\lambda_K(a_i) \approx K$, and SGD increases the likelihood of the item a_i more aggressively. As $\pi_\theta(a_i)$ reaches to a large enough value, $\lambda_K(a_i)$ goes to 0. As a result, SGD will no longer force to increase the likelihood of this item **even when $\pi_\theta(a_i)$ is still less than 1**. This in return allows the second-best item to take up some mass in the learned policy.

Figure 3 shows the policies π_θ learned with the standard (left) and top-K off-policy correction (right). We can see that with the standard off-policy correction, although the learned policy is calibrated [23] in the sense that it still maintains the ordering of items w.r.t. their expected reward, it converges to a policy that cast almost its entire mass on the top-1 item, that is $\pi(a_1) \approx 1.0$. As a result, the learned policy loses track of the difference between a slightly sub-optimal item (a_2 in this example) and the rest. The top-K correction, on the other hand, converges to a policy that has

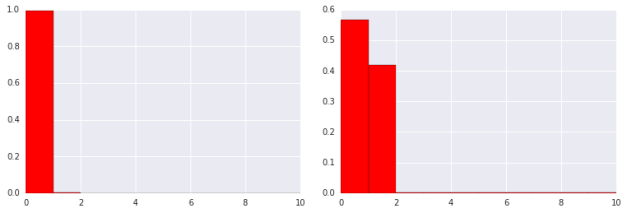

Figure 3: Learned policy π_θ (left): with standard off-policy correction; (right): with top-k correction for top-2 recommendation.

a significant mass on the second optimal item, while maintaining the order of optimality between items. As a result, we are able to recommend to users two high-reward items and aggregate more reward overall.

6.2 Live Experiments

While simulated experiments are valuable to understand new methods, the goal of any recommender systems is ultimately to improve *real* user experience. We therefore conduct a series of A/B experiments running in a live system to measure the benefits of these approaches.

We evaluate these methods on a production RNN candidate generation model in use at YouTube, similar to the setup described in [6, 11]. The model is one of many candidate generators that produce recommendations, which are scored and ranked by a separate ranking model before being shown to users on the YouTube Homepage or the side panel on the video watch page. As described above, the model is trained following the REINFORCE algorithm. The immediate reward r is designed to reflect different user activities; videos that are recommended but not clicked receive zero reward. The long term reward R is aggregated over a time horizon of 4–10 hours. In each experiment both the control and the test model use the same reward function. Experiments are run for multiple days, during which the model is trained continuously with new events being used as training data with a lag under 24 hours. While we look at various online metrics with the recommender system during live experiments, we are going to focus our discussion on the amount of time user spent watching videos, referred to as ViewTime.

The experiments presented here describe multiple sequential improvements to the production system. Unfortunately, in such a setting, the latest recommender system provides the training data for the next experiment, and as a result, once the production system incorporates a new approach, subsequent experiments cannot be compared to the earlier system. Therefore, each of the following experiments should be taken as the analysis for each component individually, and we state in each section what was the previous recommender system from which the new approach receives data.

6.2.1 Exploration. We begin with understanding the value of exploratory data in improving model quality. In particular, we would like to measure if serving a stochastic policy, under which we sample from the softmax model as described in Section 5, results in better recommendations than serving a deterministic policy where the model always recommends the K items with the highest probability according to the softmax.

We conducted a first set of experiments to understand the impact of serving a stochastic policy vs. a deterministic one while keeping the training process unchanged. In the experiment, the control population is served with a deterministic policy, while a small slice of test traffic is served with the stochastic policy as described in Section 5. Both policies are based on the same softmax model trained as in Equation (2). To control the amount of randomness in the stochastic policy at serving, we varied the temperature used in Equation (5). A lower T reduces the stochastic policy to a deterministic one, while a higher T leads to a random policy that recommends any item with equal chance. With T set to 1, we observed *no statistically significant change* in ViewTime during the experiment, which suggests the amount of randomness introduced from sampling does not hurt the user experience directly.

However, this experimental setup does not account for the benefit of having exploratory data available during training. One of the main biases in learning from logged data is that the model does not observe feedback of actions not chosen by the previous recommendation policy, and exploratory data alleviates this problem. We conducted a followup experiment where we introduce the exploratory data into training. To do that, we split users on the platform into three buckets: 90%, 5%, 5%. The first two buckets are served with a deterministic policy based on a deterministic model and the last bucket of users is served with a stochastic policy based on a model trained with exploratory data. The deterministic model is trained using only data acquired in the first two buckets, while the stochastic model is trained on data from the first and third buckets. As a result, these two models receive the same amount of training data, but the stochastic model is more likely to observe the outcomes of some rarer state, action pairs because of exploration.

Following this experimental procedure, we observe a statistically significant increase in ViewTime by 0.07% in the test population. While the improvement is not large, it comes from a relatively small amount of exploration data (only 5% of users experience the stochastic policy). We expect higher gain now that the stochastic policy has been fully launched.

6.2.2 Off-Policy Correction. Following the use of a stochastic policy, we tested incorporating off-policy correction during training. Here, we follow a more traditional A/B testing setup [6] where we train two models, both using the full traffic. The control model is trained following Equation (2), only weighting examples by the reward. The test model follows the structure in Figure 1, where the model learns both a serving policy π_θ as well as the behavior policy $\beta_{\theta'}$. The serving policy is trained with the off-policy correction described in Equation (3) where each example is weighted not only by the reward but also the importance weight $\frac{\pi_\theta}{\beta_{\theta'}}$ for addressing data bias.

During experiments, we observed the learned policy (test) starts to deviate from the behavior policy (control) that is used to acquire the traffic. Figure 4 plots a CDF of videos selected by our nominator in control and experiment according to the rank of videos in control population (rank 1 is the most nominated video by the control model, and the rightmost is least nominated). We see that instead of mimicking the model (shown in blue) used for data collection, the

[6] In practice, we use a fairly small portion of users as test population; as a result, the feedback we logged are almost entirely acquired by the control model.

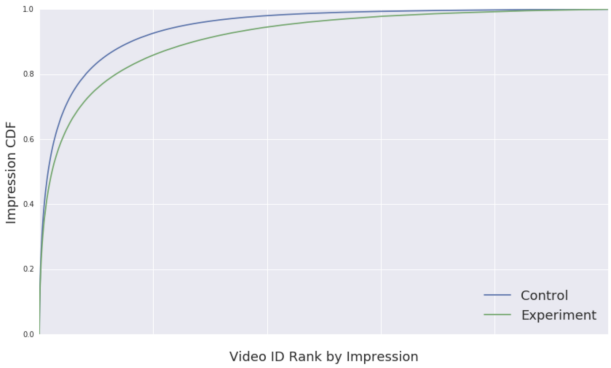

Figure 4: CDF of videos nominated in control and test population according to rank of videos in the control population. Standard off-policy correction addresses the "rich get richer" phenomenon.

test model (shown in green) favors videos that are less explored by the control model. We observed that the proportion of nominations coming from videos outside of the top ranks is increased by nearly a factor of three in experiment. This aligns with what we observed in the simulation shown in Figure 2. While ignoring the bias in the data collection process creates a "rich get richer" phenomenon, whereby a video is nominated in the learned policy simply because it was heavily nominated in the behavior policy, incorporating the off-policy correction reduces this effect.

Interestingly, in live experiment, we did not observe a statistically significant change in ViewTime between control and test population. However, we saw an increase in the number of videos viewed by 0.53%, which was statistically significant, suggesting that users are indeed getting more enjoyment.

6.2.3 Top-K Off-Policy. We now focus on understanding if the top-K off-policy learning improves the user experience over the standard off-policy approach. In this case, we launched an equivalently structured model now trained with the top-K off-policy corrected gradient update given in Equation (6) and compared its performance to the previous off-policy model, described in Section 6.2.2. In this experiment, we use $K = 16$ and capping $c = e^3$ in Equation (8); we will explore these hyperparameters in more detail below.

As described in Section 4.3 and demonstrated in the simulation in Section 6.1.2, while the standard off-policy correction we tested before leads to a policy that is overly-focused on getting the top-1 item correct, the top-K off-policy correction converges to a smoother policy under which there is a non-zero mass on the other items of interest to users as well. This in turn leads to better top-K recommendation. Given that we can recommend multiple items, the top-K off-policy correction leads us to present a better full-page experience to users than the standard off-policy correction. In particular, we find that the amount of ViewTime increased by 0.85% in the test traffic, with the number of videos viewed slightly decreasing by 0.16%.

6.2.4 Understanding Hyperparameters. Last, we perform a direct comparison of how different hyperparameter choices affect the

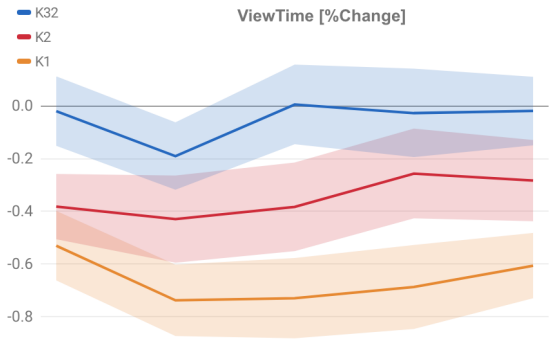

Figure 5: top-K off-policy correction with varying K.

top-K off-policy correction, and in turn the user experience on the platform. We perform these tests after the top-K off-policy correction became the production model.

Number of actions. We first explore the choice of K in the top-K off-policy correction. We train three structurally identical models, using $K \in \{1, 2, 16, 32\}$; The control (production) model is the top-K off-policy model with $K = 16$. We plot the results during a 5-day experiment in Figure 5. As explained in Section 4.3, with $K = 1$, the top-K off-policy correction reduces to the standard off-policy correction. A drop of 0.66% ViewTime was observed for $K = 1$ compared with the baseline with $K = 16$. This further confirms the gain we observed shifting from the standard off-policy correction to the top-K off-policy correction. Setting $K = 2$ still performs worse than the production model, but the gap is reduced to 0.35%. $K = 32$ achieves similar performance as the baseline. We conducted follow up experiment which showed mildly positive gain in ViewTime (+0.15% *statistically significant*) when $K = 8$.

Capping. Here we consider the effect of the variance reduction techniques on the final quality of the learned recommender. Among the techniques discussed in Section 4.4, weight capping brings the biggest gain online in initial experiments. We did not observe further metric improvements from normalized importance sampling, or TRPO [36]. We conducted a regression test to study the impact of weight capping. We compare a model trained using cap $c = e^3$ (as in production model) in Equation (8) with one trained using $c = e^5$. As we lift the restriction on the importance weight, the learned policy π_θ could potentially overfit to a few logged actions that accidentally receives high reward. Swaminathan and Joachims [43] described a similar effect of propensity overfitting. During live experiment, we observe a significant drop of 0.52% in ViewTime when the cap on importance weight was lifted.

7 CONCLUSION

In this paper we have laid out a practical implementation of a policy gradient-based top-K recommender system in use at YouTube. We scale up REINFORCE to an action space in the orders of millions and have it stably running in a live production system. To realize the full benefits of such an approach, we have demonstrated how to address biases in logged data through incorporating a learned logging policy and a novel top-K off-policy correction. We conducted extensive analysis and live experiments to measure empirically the

importance of accounting for and addressing these underlying biases. We believe these are important steps in making reinforcement learning practically impactful for recommendation and will provide a solid foundation for researchers and practitioners to explore new directions of applying RL to recommender systems.

8 ACKNOWLEDGEMENTS

We thank Craig Boutilier for his valuable comments and discussions.

REFERENCES

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. *arXiv preprint arXiv:1705.10528* (2017).

[2] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. 2014. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*. 1638–1646.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.

[4] Yoshua Bengio, Jean-Sébastien Senécal, et al. 2003. Quick Training of Probabilistic Neural Nets by Importance Sampling.. In *AISTATS*. 1–9.

[5] James Bennett, Stan Lanning, et al. 2007. The netflix prize. In *Proceedings of KDD cup and workshop*, Vol. 2007. New York, NY, USA, 35.

[6] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent Cross: Making Use of Context in Recurrent Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 46–54.

[7] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[8] Léon Bottou, Jonas Peters, Joaquin Quiñonero-Candela, Denis X Charles, D Max Chickering, Elon Portugaly, Dipankar Ray, Patrice Simard, and Ed Snelson. 2013. Counterfactual reasoning and learning systems: The example of computational advertising. *The Journal of Machine Learning Research* 14, 1 (2013), 3207–3260.

[9] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*. 2249–2257.

[10] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[11] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[12] Nathaniel D Daw, John P O'doherty, Peter Dayan, Ben Seymour, and Raymond J Dolan. 2006. Cortical substrates for exploratory decisions in humans. *Nature* 441, 7095 (2006), 876.

[13] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. 2018. Offline A/B testing for Recommender Systems. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 198–206.

[14] Ruiqi Guo, Sanjiv Kumar, Krzysztof Choromanski, and David Simcha. 2016. Quantization based fast inner product search. In *Artificial Intelligence and Statistics*. 482–490.

[15] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 173–182.

[17] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[19] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. Ieee, 263–272.

[20] How Jing and Alexander J Smola. 2017. Neural survival recommender. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 515–524.

[21] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 781–789.

[22] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32, 11 (2013), 1238–1274.

[23] Maksim Lapin, Matthias Hein, and Bernt Schiele. 2016. Loss functions for top-k error: Analysis and insights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1468–1477.

[24] Thomas Laurent and James von Brecht. 2016. A recurrent neural network without chaos. *arXiv preprint arXiv:1612.06212* (2016).

[25] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *JMLR* 17, 1 (2016), 1334–1373.

[26] Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *International Conference on Machine Learning*. 1–9.

[27] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 661–670.

[28] Jérémie Mary, Romaric Gaudel, and Philippe Preux. 2015. Bandits and recommender systems. In *International Workshop on Machine Learning, Optimization and Big Data*. Springer, 325–336.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[31] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. 2016. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*. 1054–1062.

[32] Art B. Owen. 2013. *Monte Carlo theory, methods and examples.*

[33] Doina Precup. 2000. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series* (2000), 80.

[34] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. 2001. Off-policy temporal-difference learning with function approximation. In *ICML*. 417–424.

[35] Tobias Schnabel, Paul N Bennett, Susan T Dumais, and Thorsten Joachims. 2018. Short-term satisfaction and long-term coverage: Understanding how users tolerate algorithmic exploration. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 513–521.

[36] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[37] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

[38] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[39] Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. 2010. Learning from logged implicit exploration data. In *Advances in Neural Information Processing Systems*. 2217–2225.

[40] Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*. MIT press.

[41] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[42] Adith Swaminathan and Thorsten Joachims. 2015. Batch learning from logged bandit feedback through counterfactual risk minimization. *Journal of Machine Learning Research* 16, 1 (2015), 1731–1755.

[43] Adith Swaminathan and Thorsten Joachims. 2015. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*. 3231–3239.

[44] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miro Dudik, John Langford, Damien Jose, and Imed Zitouni. 2017. Off-policy evaluation for slate recommendation. In *Advances in Neural Information Processing Systems*.

[45] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 17–22.

[46] Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

[47] Philip Thomas and Emma Brunskill. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*. 2139–2148.

[48] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[49] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *Proceedings of the tenth ACM international conference on web search and data mining*. ACM, 495–503.

[50] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. Deep Reinforcement Learning for Page-wise Recommendations. *arXiv preprint arXiv:1805.02343* (2018).