User Response Models to Improve a REINFORCE Recommender System

Minmin Chen, Bo Chang, Can Xu, Ed H. Chi
Google, Inc.
Mountain View, CA
minminc,bochang,canxu,edchi@google.com

ABSTRACT

Reinforcement Learning (RL) techniques have been sought after as the next-generation tools to further advance the field of recommendation research. Different from classic applications of RL, recommender agents, especially those deployed on commercial recommendation platforms, have to operate in extremely large state and action spaces, serving a dynamic user base in the order of billions, and a long-tail item corpus in the order of millions or billions. The (positive) user feedback available to train such agents is extremely scarce in retrospect. Improving the sample efficiency of RL algorithms is thus of paramount importance when developing RL agents for recommender systems. In this work, we present a general framework to augment the training of model-free RL agents with auxiliary tasks for improved sample efficiency. More specifically, we opt to add additional tasks that predict users' immediate responses (positive or negative) toward recommendations, i.e., user response modeling, to enhance the learning of the state and action representations for the recommender agents. We also introduce a tool based on gradient correlation analysis to guide the model design. We showcase the efficacy of our method in offline experiments, learning and evaluating agent policies over hundreds of millions of user trajectories. We also conduct live experiments on an industrial recommendation platform serving billions of users and tens of millions of items to verify its benefit.

KEYWORDS

Recommender Systems, Auxiliary Tasks, User Response Models, Reinforcement Learning

ACM Reference Format:
Minmin Chen, Bo Chang, Can Xu, Ed H. Chi. 2021. User Response Models to Improve a REINFORCE Recommender System. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437963.3441764

1 INTRODUCTION

Recommender systems are heavily relied on by many commercial recommendation platforms to serve their user base. The recommendation algorithms powering these systems have evolved drastically over the years, starting from the simple collaborative filtering based approaches [24, 34], to supervised learning approaches based on user feedback [7, 50], and more recently reinforcement learning (RL) techniques have been gaining a lot of attraction in the field of recommendation research [3, 18, 52, 53]. RL focuses on building agents that can take actions in an environment to maximize some definition of long-term reward [42]. Under the recommendation settings, this translates into building agents that can interact with users so as to optimize their long-term experience on the recommendation platform.

Compared with supervised learning approaches that treat the immediate user feedback (e.g., clicks, dwell time, likes, share) observed on recommended items as the ground truth, the RL (or bandits) formulation naturally fits the partial feedback setup in real-world recommender systems. In other words, we only observe feedback on items recommended, but not the others, which could potentially be more rewarding for users to consume. The sequential planning aspect of RL also offers opportunities for the recommender systems to move beyond optimizing for immediate user feedback and toward long-term user enjoyment of the platform.

Despite these attractions, realizing RL in industrial-scale recommender systems has been challenging. In contrast to classic RL applications, recommender systems, especially industrial ones, have to serve an extremely large user base with dynamic interests and a huge item corpus. That is, it has to scale to an extremely large state and action space that is in the orders of millions or billions. The extreme sparsity of user feedback amongst this large state and action space further exacerbates the problem. Each user has only been exposed to a small fraction of the item corpus, an even smaller fraction of which actually receives (positive) user feedback. Increased computational power has enabled RL algorithms in classical applications, such as robotics and games, to rely on simulation or self-play [39] to generate a large number of learning trajectories to combat sample inefficiency. RL for recommendation applications on the other hand is distinct in its challenge in coming up with simulated user interaction trajectories that are realistic. As a result, interaction and feedback data collected on the recommendation platform are the sole source of data we can rely on for learning.

Improving RL algorithms to fully utilize the available user feedback is therefore critical when building RL agents for recommendation platforms. Different approaches have been proposed in the RL literature to address sample inefficiency, which can roughly be grouped into two categories: 1) introducing auxiliary tasks based on agents' sensory data to directly augment and speed up the learning of model-free RL agents [15, 19, 37, 41]; 2) learning the dynamics (world models) [13, 14] and reward function, and use them to plan

WSDM '21, March 8–12, 2021, Virtual Event, Israel.

ACM ISBN 978-1-4503-8297-7/21/03.
https://doi.org/10.1145/3437963.3441764

for model-based RL agents [21] or rollout to collect additional trajectories to train model-free RL agents. Our approach falls closer to the first category. Different from existing works that mainly target applications of RL involving sensory data, such as images or videos, we focus on the design of auxiliary tasks for recommendation settings. More specifically, we offer the following contributions:

- **A Framework for Introducing Auxiliary Tasks**: We present a general framework to improve sample efficiency of model-free RL algorithms for recommendation applications through augmenting the policy learning with auxiliary tasks.
- **User Response Models**: We introduce auxiliary tasks of predicting users' responses (positive or negative) toward recommendations, i.e., user response modeling, to improve the state and action representations of model-free RL agents.
- **Practical Lessons**: We share our experience in designing these user response models, e.g., architecture choices, training setup, targeted treatment, and computational considerations.
- **Analysis Tool to Guide Development**: We offer an analysis tool based on the gradient correlation between policy learning and the auxiliary tasks to guide the decision of various design choices.
- **Offline and Live Experiments**: We conduct large-scale offline experiments to compare our approach against a state-of-the-art REINFORCE agent and show improvement. We also test the proposed method in live experiments on a commercial recommendation platform serving billions of users and millions of items and showcase the benefit of our approach in improving long-term user enjoyment of the platform and its scalability.

2 RELATED WORK

Reinforcement Learning for Recommender Systems. Classic reinforcement learning approaches [42] can be roughly grouped into model-based and model-free variants, with the latter one further divided into value-based approaches such as Q-learning [48] and policy-based ones such as REINFORCE [49]. Modern RL approaches combine these techniques with high-capacity function approximators, i.e., deep neural networks, and increased computational power, achieving immense success in various domains [14, 28, 40]. The promise of deep RL has sparked a lot of interest in utilizing it for recommender systems. Shani et al. [38] first brought up a Markov decision process (MDP) formulation of recommender systems and proposed a model-based approach. Zheng et al. [55] explored DQN for news recommendation. Dulac-Arnold et al. [9] focused on the large action space problem by modeling the state in the same continuous item embedding space and selecting the items via nearest neighborhood search. Liu et al. [29] studied an actor-critic approach for recommendation. Several recent works [3, 18, 54] further extend deep RL to set recommendation problems. Despite exciting research advances, the success of RL in real-world recommendation applications is still rare. Chen et al. [3] managed to scale a REINFORCE agent to handle an extremely large action space by learning on historical user interaction data and applying off-policy correction to address the batch learning effect. Hu et al. [17] framed learning to rank as an MDP and learned a ranking policy through deterministic policy gradient (DPG) and tested the approach on a commercial platform.

Auxiliary Learning. It is well known that neural networks require a large amount of training data to achieve state-of-the-art performance. The problem is even more pronounced in deep RL due to off-policy training. One approach to improving the sample efficiency of neural networks is to leverage auxiliary tasks to help construct useful representations. In addition to the primary task, one or more auxiliary objectives are simultaneously optimized during training. Note that only the primary task is of interest, and the purpose of auxiliary tasks is merely to learn better representations for the primary task; this is different from multi-task learning.

Auxiliary learning has been widely studied in various areas, including sequence modeling [47], reinforcement learning [15, 19, 37, 41], computer vision [31, 51], speech recognition [46], and meta-learning [30]. Ideally, auxiliary tasks should be well aligned with the primary one so that they can help the primary task. It is however non-trivial to design such tasks. Du et al. [8] propose to use the cosine similarity between gradients of tasks as an adaptive weight to detect when an auxiliary loss is helpful to the primary one. We adopt the analysis in making different design choices when developing our framework.

Continual Learning. Another line of work that motivates our auxiliary task design is the continual learning literature [23, 32, 33], where the agents transfer learning from easier tasks to harder ones without catastrophic forgetting. While the long-term planning perspective of RL is attractive in finding policies to optimize users' experience in the long run, it also introduces a lot of uncertainty into the learning of the agents' policies. We regard the transition from predicting user's immediate response toward long-term planning as one instance of continual learning.

3 A REINFORCE RECOMMENDER SYSTEM

We build our work on top of the REINFORCE recommender system introduced in [3]. The recommendation problem is translated into a Markov Decision Process (MDP)[1] $(\mathcal{S}, \mathcal{A}, \mathbf{P}, R, \rho_0, \gamma)$. Here $\mathcal{S}$ is a continuous latent state space describing the user state and context, $\mathcal{A}$ is 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 state s, ρ_0 is the initial state distribution, and γ is the discount factor for future rewards.

A recommender agent is built by parameterizing and learning a softmax policy

$$\pi_\theta(a|\mathbf{s}) = \frac{\exp(\mathbf{u}_s^\top \mathbf{v}_a / T)}{\sum_{a' \in \mathcal{A}} \exp(\mathbf{u}_s^\top \mathbf{v}_{a'} / T)}, \tag{1}$$

which defines a distribution over the item corpus $\mathcal{A}$ conditioning on the user state $s \in \mathcal{S}$. Here $\mathbf{u}_s$ stands for the learned latent representation of the user state, $\mathbf{v}_a$ is the item representation, and $T > 0$ is the temperature that adjusts the entropy of the learned policy. The policy parameters θ are learned using REINFORCE [49] so as

[1]The states are only partially observable, making it a Partially Observable MDP (POMDP). For simplicity, we will leave the notations as is.

to maximize the expected cumulative reward over the interaction trajectories,

$$\max_{\theta} \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right], \text{ where } R(\tau) = \sum_{t=0}^{|\tau|} r(s_t, a_t). \quad (2)$$

Note that the expectation is taken over the trajectories $\tau = (s_0, a_0, s_1, \ldots)$ obtained by acting according to the latest learned policy: $s_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot|s_t)$, and $s_{t+1} \sim \mathbf{P}(\cdot|s_t, a_t)$.

Batch Reinforcement Learning. As noted above, reinforcement learning is designed as an online learning paradigm in the first place [42]. The agent's policy is learned by iterating between interacting with the environment to collect experiences according to the latest learned policy, and improving the learned policy using the collected experiences. Unfortunately, in many real-world applications of RL, interacting with the environment in real-time is not feasible. In recommender systems, for example, we instead deploy a learned agent with a fixed policy and use it to interact with the users and collect experiences for a period of hours or days. The collected trajectories are then used to perform many updates before a new version of the policy is deployed. As a result, the collected trajectories can follow a very different distribution than that of trajectories would have been generated from the current policy. Realizing the interactive nature of online RL has been hindering its wider adoption, researchers spent a great amount of effort to bring these techniques offline, which is commonly referred to as the batch or off-policy reinforcement learning [2, 4, 10, 11, 22, 25–27, 43, 45].

Sample Inefficiency under Off-Policy RL. A long-established technique for addressing distribution shift, or off-policy training under RL terminology, is to apply an importance sampling correction. Naively, the expected cumulative reward defined in eq. (2) can be approximated using trajectories sampled from a different distribution β, commonly referred to as the behavior policy, as

$$\mathbb{E}_{\tau\sim\beta}\left[\frac{\pi_\theta(\tau)}{\beta(\tau)} R(\tau)\right].$$

The variance of the this estimator unfortunately grows exponentially with respect to the length of the trajectories $|\tau|$. Chen et al. [3] reduce the variance by taking a first-order approximation to $\pi_\theta(\tau)/\beta(\tau)$ [1], which results in a *weighted cross-entropy loss* as follows:

$$\ell_{RL}(\theta) = -\sum_{s_t\sim d_t^\beta(s), a_t\sim\beta(\cdot|s_t)} \left[\frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)} R_t \log \pi_\theta(a_t|s_t)\right]. \quad (3)$$

Here $\beta(\cdot|s)$ denotes the action distribution conditioning on state s in the collected trajectories, $d_t^\beta(s)$ is the discounted state visitation probability under the behavior policy [27], and $R_t = \sum_{t'=t}^{|\tau|} \gamma^{t'-t} r(s_{t'}, a_{t'})$ is the discounted cumulative reward starting from time t. This importance weight is further adapted to accommodate the set recommendation setup. We refer the interested readers to [3] for more details.

Even with the simplification, the loss prescribed in eq. (3) can still suffer from high variance when the learned policy π_θ deviates from the behavior policy β. Existing works mostly focus on reducing the variance by constraining the learned policy to be close to the behavior policy [35, 36]. The constraint, however, also limits one's ability to find the optimal policy that maximizes the long-term reward.

4 METHOD

We instead focus on improving the sample efficiency of the base REINFORCE recommender agent by augmenting the training of the agent policy with auxiliary objectives. We start by laying out the overall framework to minimally change the REINFORCE training with auxiliary losses, and then specify several design choices we made in coming up with User response modeling for RL (URL).

4.1 Auxiliary Tasks to Improve Sample Efficiency

Figure 1 depicts the overall framework. The blocks shown in solid lines are the main components of a REINFORCE recommender agent. The user representation block is in charge of learning the latent representation of the user state and context, usually based on observations of users' historical activities on the recommendation platform. The action representation block learns embeddings of the items available for recommendation. Given the state and action representations, the REINFORCE agent learns a policy $\pi_\theta(\cdot|s)$ based on the state-action-reward tuples (s_t, a_t, R_t) collected in the user trajectories as in eq. (3). As mentioned above, the biggest challenges in applying RL for recommender systems lie in the large state and action space and the extreme sparsity of user feedback with respect to the vast state and action space. Off-policy training further exacerbates the problem since as the learned policy deviates from the behavior policy used to acquire the trajectories, the large variance in the importance weights reduces the effective sample size for learning [6].

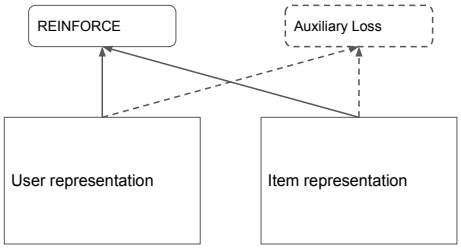


Figure 1: A framework to improve sample efficiency of model-free RL through introducing auxiliary losses.

To combat the sparsity of user feedback and sample inefficiency of off-policy RL, we propose to add the blocks shown in dashed lines representing the auxiliary training objectives to supplement the learning of the policy parameters, in particular, the user state and action representations. Using auxiliary tasks to improve representation learning for RL has been extensively explored in the literature [15, 19, 37, 41]. Different from most existing work, in which the auxiliary tasks are added to improve the state representation learned from agents' sensory data, such as image pixels, we allow the auxiliary objectives, trained jointly with the main REINFORCE objective, to influence both the state and action representations.

4.2 User Response Modeling

The most critical design we have to decide on is the auxiliary objectives. Unsupervised tasks, such as reconstruction-based auxiliary losses [15, 19] and contrastive learning losses [37, 41] have been widely adopted in the RL literature for applications such as games and robotics, in which the environment state and state transition are captured in high-dimensional images or videos. It is however less straight-forward to find counterparts of these unsupervised learning tasks for recommendation problems. For a recommender agent, the environment it interacts with is the user. The agent does not observe the full user states, which can depend on a lot of exogenous factors; instead, it has to infer the user state from the observations, e.g., the user's historical activities on the platform. Without a fully-observed state, designing reconstruction-based losses becomes less obvious.

We instead resort to supervised learning tasks, which ensure the state and action representation of the agent is able to predict the user's response when exposed to a recommendation. We restrict the label of the auxiliary task to the **immediate** user response for two reasons: 1) to circumvent the challenge of off-policy training. The expected long-term reward $\mathbb{E}_{\tau\sim\pi}[R(\tau)]$ depends on the deployed policy; the immediate user response $\tilde{r}(s,a)$ on the other hand is independent of the deployed policy and only depends on the current state and action; 2) to introduce relatively easier tasks for the agent to progress toward the long-term objectives. Long-term planning is more difficult as the agent has to reason about not only the immediate outcome of its action, but also its long term impact, which can be very noisy to start with. We assist the agent in better planning for the future by first making sure its state and action representation can predict users' immediate response, i.e., the immediate outcome of its action. Note that $\tilde{r}(s,a)$ can be identical to or different from the immediate reward $r(s,a)$ used to train the main REINFORCE head as defined underneath eq. (3). The task can learn to predict different engagement signals such as click, dwell time, or post-engagement signals, such as like, dislike, share, or survey responses.

辅助任务不依赖于目标策略，这样对主任务的帮助作用/信息增益会更大一些，否则跟主任务显得有些重复

We name our method of augmenting model-free RL training with user response modeling as *User response modeling for RL*, or URL in short.

4.3 Architecture Overview

Figure 2 presents one instantiation of URL. The blocks in solid lines represent the main components of the REINFORCE agent. It captures the user state and state transition through a Recurrent Neural Network (RNN) [5, 16], going through observations of user historical activities on the platform. The output of the RNN, concatenated with label context that captures the context under which a recommendation is made, for example time, device, page information, forms the user state representation. The latent state representation is then sent through multiple layers of ReLU units and a linear projection down to the same dimension as the action embeddings $\mathbf{v}_a$. The inner product of the state and action representations $\mathbf{u}_s^\top \mathbf{v}_a$ forms the logits of the softmax policy defined in eq. (1).

The blocks in dashed lines correspond to the components of the auxiliary task. The auxiliary objective takes the concatenation of the user state and the context embeddings, send them through a linear projection layer to project down to again the same dimension as the action embeddings. The auxiliary objective ensures the projected

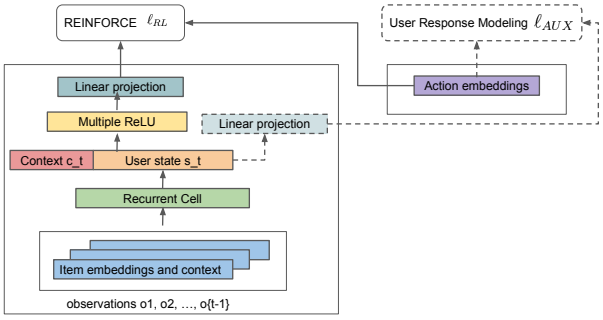


Figure 2: User response models to augment the training of the REINFORCE agent.

state $\mathbf{u}'_s$, along with the action embedding $\mathbf{v}_a$ can predict the user response toward the recommendation:

$$\ell_{AUX}(\theta) = \sum_{s_t\sim d_t^\beta(s), a_t\sim\beta(\cdot|s_t)} \ell(\mathbf{u}_{s_t}'^\top \mathbf{v}_{a_t}, \tilde{r}(s_t,a_t)). \quad (4)$$

The loss function ℓ needs to be properly chosen for different response signals. Here we detail two instantiations of the loss for different response signals. Let $h(s,a) = \mathbf{u}_s'^\top \mathbf{v}_a$ [2] and $p(s,a) = \text{sigmoid}(h(s,a))$. When the response signal is binary such as click, i.e., $\tilde{r}(s,a) = 1$ if the user clicks on the recommendation and 0 otherwise, we use the binary cross-entropy loss:

$$\ell_{AUX}(\theta) = \tilde{r}(s_t,a_t)\log p(s_t,a_t) + (1-\tilde{r}(s_t,a_t))\log(1-p(s_t,a_t)). \quad (5)$$

When the response signal is a real number with potentially large values, such as dwell time, we use the Huber loss to avoid overfitting to outliers:

$$\ell_{AUX}(\theta) = L_{\text{Huber}}(h(s_t,a_t) - \tilde{r}(s_t,a_t)). \quad (6)$$

Note in this case, the auxiliary loss is learned only on state-action pairs with a positive response $\tilde{r}(s,a)$.

The auxiliary head shares the majority of its parameters with the main REINFORCE head. As shown in Figure 2, all the embeddings of historical events, the RNN parameters, as well as the embeddings of the actions are shared between the REINFORCE and the auxiliary objectives. The auxiliary objective does have its own parameters, the linear projection in this specific instance, to ensure it can properly transform the hidden states from the RNNs, the context features, and the action embeddings to optimize a different objective than that of the main REINFORCE head. The reasons to use a simpler transformation (a single linear layer) for the auxiliary task than that of the main REINFORCE (multiple ReLU units and a linear projection) are: 1) the immediate response is easier to predict than optimizing for long-term reward as in the REINFORCE objective; 2) by limiting extra parameters specific to the auxiliary task, we allow more influence of the auxiliary objective on the shared

[2] Other forms to come up with $h(s,a)$ from the user and action representations are possible. We compare them in Section 5.

parameters. As a result, we learn better state and action representations for the REINFORCE agent, which is the ultimate goal. The shared parameters as well as task-specific ones are trained jointly to minimize the combination of the REINFORCE and the auxiliary losses:

$$\min_{\theta} \ell_{RL}(\theta) + \lambda \ell_{AUX}(\theta). \tag{7}$$

Here θ denotes the shared and task-specific parameters, and λ is a scalar to control the influence of the auxiliary tasks.

4.4 Training Setup

Figure 3 shows how we split a user trajectory to train the main REINFORCE head and the auxiliary task. For each user, we have access to a sequence of historical interactions $\tau = \{(s_t, A_t, a_t, r_t, \tilde{r}_t) : t = 0, \ldots, T\}$. We use A_t to denote all the items that have been recommended to the user at time t, with $a_t \in A_t \cup \{\text{null}\}$ denoting the item the user interacted with, r_t the immediate reward used in REINFORCE training, and $\tilde{r}_t$ the user response used in the auxiliary task training. Note that $a_t = \text{null}$ indicates that the user did not interact with any item from A_t, and $r_t = \tilde{r}_t = 0$ in this case. We follow the convention in [3] and assign zero immediate and long-term rewards to any items in A_t that did not result in a direct user interaction. $R_t(s_t, a) = 0, \forall a \in A_t \backslash \{a_t\}$ effectively zeros out the gradient on these state-action pairs. As a result, items in $A_t \backslash \{a_t\}$ are excluded from the REINFORCE objective[3] defined in eq. (3).

The auxiliary loss(es) defined in eq. (4), on the other hand, can train on all recommended items, $(s_t, a, \tilde{r}_t), \forall a \in A_t$, for example in the case of predicting click. In the case of dwell time prediction, we choose to learn the auxiliary loss only on (s_t, a_t), i.e., the same interactions as used in the REINFORCE objectives. One might wonder whether the auxiliary loss can still help in this case. We argue that it can serve as an easier task to bootstrap the learning of the agent's policy to optimize for the long-term reward, especially when $\tilde{r}(s, a)$ aligns with the immediate reward $r(s, a)$.

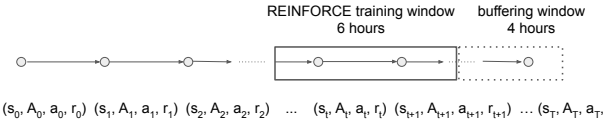


Figure 3: Training windows for the main REINFORCE head and the auxiliary losses.

One unique aspect of the recommendation problem is that the user preferences shift over time, as well as the item corpus. To keep the agent up to date with the latest user dynamics and item corpus, we limit the training window of the main REINFORCE head (eq. 3) to (s_t, a_t, R_t) tuples within the last 6 hours of the trajectories, as shown in the middle session of the figure, with a buffering window of 4 hours at the end of the trajectory to compute the cumulative discounted reward R_t. The auxiliary loss (eq. 4) on the other hand trains on any item $(s_t, a, \tilde{r}(s_t, a_t)), a \in A_t$ on the trajectory.

[3]We still use these items along those user interacted with to capture the user state.

4.5 Low-Activity Users

Even though we strive to keep an equal length[4] of trajectories for users, the entire trajectory can span months for some users while for others it takes less than a day. As a result, within the 6 hours of training window for the main REINFORCE head, different users can have very different numbers of tuples (s_t, a_t, R_t) available to train the agent's policy. Figure 4 shows histograms of the number of training labels for different users over the training steps (z-axis). While some users can have hundreds of positive interactions within the hours, more than 50% of users have less than five to contribute to the loss of the REINFORCE head. As a result, the agent's policy is more apt to highly active users than the less active ones.

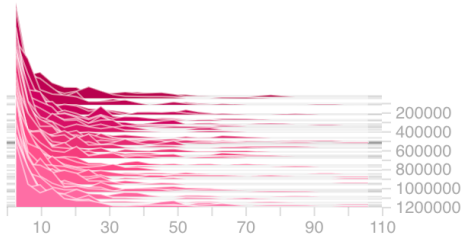


Figure 4: Number of training labels per user for the main REINFORCE head.

To compensate for that, we choose to activate the auxiliary objective only for users who are less active. To determine a user's activity level, we look back in his/her activities in the past two weeks, and de-activate the auxiliary objective if the user has been coming to the recommendation platform everyday.

4.6 Computational Complexity

Another consideration we take into account when designing the auxiliary tasks is the computational cost of computing these auxiliary losses. The main REINFORCE loss as defined in eq. (3) is extremely expensive to compute for large action space applications, as the softmax policy (eq. 1) requires computing the partition function over the entire action space. We follow [3] and approximate it using sampled softmax [20]. Given the vast action space, we do find the number of samples used impacts the approximation accuracy as well as the quality of the learned policy significantly. We end up using 20,000 negative samples (a') for each positive action (a_t) to approximate the partition function. The auxiliary losses as defined in eq. (5) and (6), on the other hand, do not depend on the size of the action space. It, therefore, takes only a fraction of the compute compared with the main REINFORCE head.

5 ANALYSES

In this section, we provide an analysis tool based on gradient information to guide the design of various modeling choices used in our experiments.

5.1 Gradient Similarity Between Tasks

One central question in auxiliary learning is how to design the auxiliary tasks; ideal auxiliary tasks should be related to the primary

[4]For very inactive users, their overall trajectory length can be shorter.

task, but not too aligned and therefore become redundant. Practitioners often rely on domain knowledge to design such auxiliary tasks. There is however no guarantee that they do not hurt the primary task.

A more principled way to measure task similarity is the cosine similarity of gradients between tasks [8, 30] with respect to model parameters. Given two tasks and their corresponding losses $\ell_1, \ell_2 \in \mathbb{R}$, and (a subset of) model parameters $\eta \in \mathbb{R}^d$ that contribute to both losses, the "alignment" between ℓ_1 and ℓ_2 is measured by the cosine similarity between their gradients with respect to η:

$$\text{sim}(\ell_1, \ell_2, \eta) := \frac{(\mathrm{d}\ell_1/\mathrm{d}\eta)^\top (\mathrm{d}\ell_2/\mathrm{d}\eta)}{\|\mathrm{d}\ell_1/\mathrm{d}\eta\|_2 \cdot \|\mathrm{d}\ell_2/\mathrm{d}\eta\|_2}. \tag{8}$$

Intuitively, if two tasks are aligned, their gradients should reinforce each other and geometrically form an acute angle; in other words, their cosine similarity should be positive. In the extreme case, if they are perfectly aligned and have a cosine similarity of 1, then one of the tasks is redundant. Therefore, an ideal cosine similarity should lie in the interval $(0, 1)$. This similarity measure can thus be used to guide the design of auxiliary tasks. In other words, we would like to find auxiliary losses that have a positive gradient similarity with the primary task.

5.2 Comparison with Alternative Architectures

Besides the model architecture presented in Section 4.3, we also consider a few alternatives and apply the gradient similarity analysis to make comparisons. The following design options are considered:

(1) **URL**: the original model architecture shown in Figure 2.
(2) **URL with separate ReLU layers**: we increase the number of parameters specific to the auxiliary task by adding multiple ReLU layers between the user states and context (orange and red blocks in Figure 2), and the linear projection layer (cyan block in Figure 2) for the URL head. The number of layers and hidden units is the same as the ReLU layers for the main loss (yellow block in Figure 2).
(3) **URL with concatenation**: instead of using the inner product between the latent state representation and the action embedding to form the prediction $h(s_t, a_t) = \mathbf{u}_{s_t}'^\top \mathbf{v}_{a_t}$ as in eq. (4), we concatenate them $(\mathbf{u}_{s_t}'^\top, \mathbf{v}_{a_t}^\top)$ and pass it through a two-layer ReLU network to get the scalar prediction $h(s_t, a_t)$.

For all three architectures, we use the dwell time as the target response signal $\tilde{r}(s, a)$ and apply the Huber loss as in eq. (6).

Table 1: Mean Average Precision (MAP@1) weighted by discounted cumulative reward for different architectures.

URL	URL w/ sep ReLU	URL w/ concat
0.061	0.059	0.057

Figure 5 illustrates the cosine similarity $\text{sim}(\ell_{RL}, \ell_{AUX}, \mathbf{v}_a)$ between ℓ_{RL} and ℓ_{AUX} with respect to the item representations $\mathbf{v}_a$ in eq. (1) and (4). In other words, it measures the alignment of the auxiliary and primary losses in learning the item representations and can be used to guide the architectural design. All architectures are trained for 1,000,000 steps[5]. Since the experiments are cold-started and the gradient similarity is noisy at the beginning, we ignore the first 100,000 training steps in Figure 5.

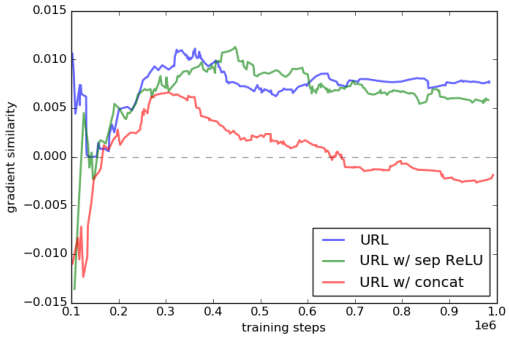


Figure 5: Gradient similarity between the primary loss ℓ_{RL} and auxiliary loss ℓ_{AUX} with respect to softmax weights $\mathbf{v}_a$.

As shown in the figure, the gradient similarity is positive for **URL** and **URL with separate ReLU layers**, suggesting the primary loss ℓ_{RL} and the auxiliary losses ℓ_{AUX} are well aligned. For **URL with concatenation**, it begins with a positive gradient similarity and later gradually decays to zero, suggesting the auxiliary loss is no longer aligned with the main head later on. Moreover, Figure 5 suggests that **URL** has the best alignment. This is further supported by Table 1, where we use the Mean Average Precision (MAP@1) weighted by discounted cumulative reward ((detailed in Section 6.1)) as an offline metric to compare the three architectures. The **URL** architecture has the highest final MAP@1 among the three architectures in consideration, which is consistent with the gradient similarity analysis.

6 OFFLINE AND LIVE EXPERIMENTS

To measure the effectiveness of our method, we test it offline on a large-scale dataset containing hundreds of millions of users and tens of millions of items, with billions of interaction/feedback between them. We also verify our approach in live experiments on a commercial recommendation platform serving billions of users.

6.1 Offline Experiments

6.1.1 Dataset. We extract hundreds of millions of user trajectories from a commercial recommendation platform. Each trajectory contains a list of user interactions up to the point of training, $\tau = \{(s_t, A_t, a_t, r_t, \tilde{r}_t) : t = 0, \ldots, T\}$, as described in Section 4.4. The lengths of trajectories between users can vary. We keep at most 500 historical pages with at least one positive user interaction (nonzero r_t) for each user. Among the collected trajectories, we hold out 1% for evaluation. We restrict our action space (item corpus) to the most popular 10 million items on the platform. Our goal is to build a recommender agent that can choose among the 10 million corpus the next item for users to consume so as to maximize the cumulative long-term reward.

[5]We restrict this set of experiments to train and evaluate on low-activity users only to get better resolution into the gradient similarity between the two tasks.

6.1.2 Setup. We compare the base REINFORCE agent, and the REINFORCE agent trained with three auxiliary loss setups: 1) a single auxiliary loss predicting whether or not the user will click on the recommended item (URL, click); 2) a single auxiliary loss predicting how long the user will interact with the recommended item (URL, dwell time); 3) a combined auxiliary loss of both tasks (URL, combined).

Recommender systems operate under the partial feedback setup, in which we only observe feedback on items that have been recommended by the deployed policy, but not others. As a result, the off-policy effect has to be accounted for in evaluation in order to estimate the performance of a new policy when deployed. Offline (off-policy) evaluation of a given policy is a challenging problem by itself. Existing works mostly leverage inverse-propensity scores, with various variance control techniques such as capping or normalization in order to reduce the variance in the off-policy estimate [12, 43–45]. Even with these variance control techniques, off-policy evaluation often results in an estimate that is noisier than desired.

We instead focus our offline experiment on comparing our proposal to the baseline approach under the "supervised learning" setup, while relying on live experiments for further validation. Here we treat the item that the user interacted with as the ground truth, ignoring the effect of partial feedback. We do not apply the off-policy correction when learning the policy either. As a result, the task becomes predicting which item the user will interact with next, with each example weighted by the long-term reward. This setup avoids the high variance in the learning and evaluation of the policies caused by the inverse-propensity weighting. Even though these experiments are less accurate in predicting live experiment results since they ignore the partial feedback (off-policy) effect, the reduced variance can offer us better resolution in understanding if introducing these auxiliary tasks helps the agent learn better state and action representations, so as to accurately predict which item the user will interact with next.

6.1.3 Metrics. The main offline metric we focus on is the Mean Average Precision (MAP@K) weighted by discounted cumulative reward R_t. In other words, whether the item the user chose to interact with next appear among the top-K result returned by the agent. For each interaction within the 6-hour window (Section 4.4) in the holdout set, we compute the MAP@K and weight it by the discounted cumulative reward this interaction leads to.

6.1.4 Results. We now present the series of experiments we conduct offline to measure the sensitivities of our method toward the different hyper-parameters as well as the effects of introducing different auxiliary tasks. We use the auxiliary task of predicting user dwell time to study the sensitivity of hyper-parameters.

Figure 6 studies the effect of the λ hyper-parameter trading-off the importance of the primary and the auxiliary task. For these experiments, we fix the training window of the auxiliary loss to be the same as the main REINFORCE loss. As shown in Figure 6, adding the auxiliary task does improve the accuracy over the base REINFORCE agent. However, as we increase the weight λ from 0.1 to 10, we can see a degrade in the performance of URL, suggesting that focusing too much on the myopic objective does interfere with the learning of the agent's policy to optimize for the cumulative long-term reward.

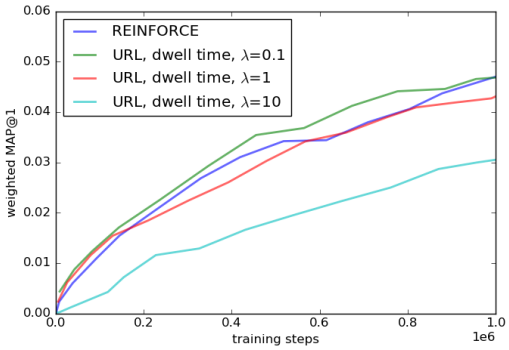


Figure 6: The effect of the λ hyper-parameter to trade off the importance of the main REINFORCE head and the auxiliary task in eq. (7).

Figure 7 studies the effect of different training window lengths used in the auxiliary task of predicting user dwell time. While keeping the training window of the auxiliary task consistent with that of the main REINFORCE head (6 hours) already improves the weighted MAP@1, we do observe even better performance when the training window is extended (2 and 7 days).

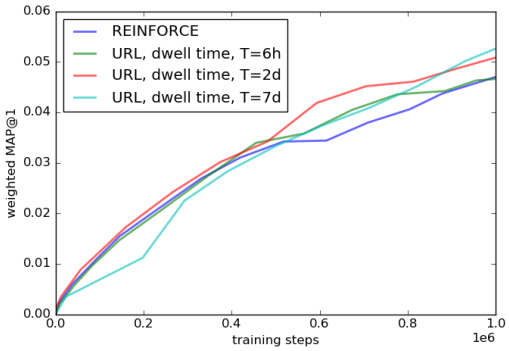


Figure 7: The effect of different training window lengths on the auxiliary task.

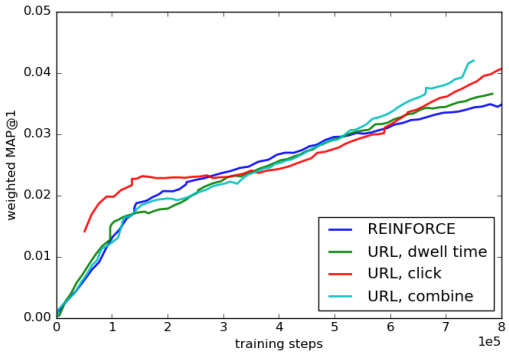
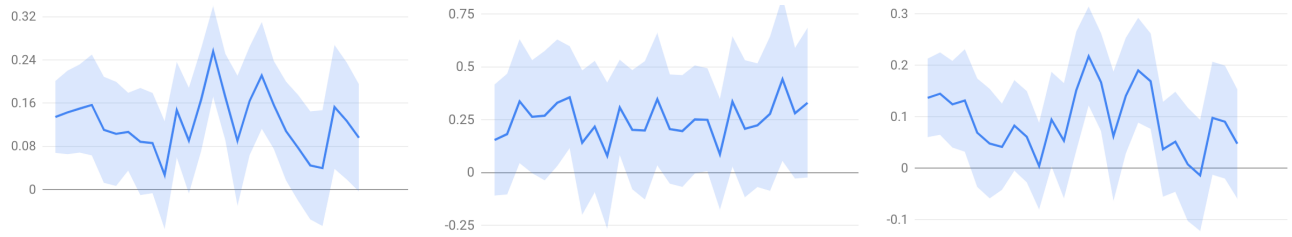


Figure 8: Comparison of the base REINFORCE agent and URL with different auxiliary tasks.



(a) Global user base **(b) Low-activity users** **(c) High-activity users**

Figure 9: Comparison of the base REINFORCE agent and URL in a live system.

Figure 8 summarizes the comparison between the base REINFORCE agent and the REINFORCE agent trained with different auxiliary tasks [6]. To simplify the comparison, we set the λ hyperparameter for all three setups to 1. As shown in the figures, introducing auxiliary tasks leads to higher weighted MAP@1 compared with the baseline. The auxiliary task of predicting the click performs slightly better than the one predicting dwell time, while combining leads to the highest performance in the end. The gradient similarity analysis tool presented in Section 5 also suggests that the click prediction auxiliary task has a higher gradient similarity with the main task than the dwell time prediction task.

6.2 Live Experiments

We conduct a series of A/B experiments in a live system serving billions of users to measure the benefits of our approach. The REINFORCE agent is built to retrieve hundreds of candidates from a corpus of 10 million items upon each user request. The retrieved candidates, along with those returned by other sources, are scored and ranked by a separate ranking system before showing the top results to the user. The REINFORCE agent is trained as prescribed in eq. (3), with the off-policy correction adapted to the set recommendation setting as suggested in [3]. The long-term reward R_t captures the discounted cumulative sum of user feedback $r(s, a)$ over the trajectories. The control runs a REINFORCE agent trained with only the RL objective, and the experiment runs a REINFORCE agent with the additional auxiliary task to predict user dwell time $\tilde{r}(s, a)$ on the recommended items. Experiments are run for a month, during which both the control and experiment models are trained continuously with new interaction and feedback being used as training data. We focus our discussion on a metric capturing users' overall enjoyment.

Figure 9 summarizes the live experiment results. We can see that our method improves over the baseline method on the global user base, leading to a +0.12% improvement with a 95% confidence interval of [+0.07%, +0.18%]. When we zoom in to examine the improvement over different user slices, we can see significantly higher improvement on the low-activity user slice (+0.26%) compared with the high-activity user slice (+0.09%). The results are aligned with the motivation of our design.

We also run another baseline experiment in which we extend the training window of the base REINFORCE agents to 7 days rather than the 6 hours used in the control setup. We find naively extending the training window of the REINFORCE agent leads to degraded performance (−0.12% with a 95% confidence interval [−0.15%, −0.09%] in the discussed metric). We believe two factors are contributing to the worse performance: 1) the extension of the training window causes the training to slow down significantly. The model turnaround time is doubled, and serving a less-fresh model/policy leads to worse performance; 2) user preferences are constantly changing on the platform, and learning based on outdated interaction data leads to worse performance. This set of experiments demonstrates that our proposed method improves over the base REINFORCE agent not simply because of additional learning signals, but also carefully designed auxiliary tasks and model architecture.

[6] We saved more checkpoints during training for this set of experiments to give more resolution into the comparison between different auxiliary tasks. As a result, Figure 8 appears more noisy than Figures 6 and 7.

7 CONCLUSIONS

We present a general framework to improve the sample efficiency of a state-of-the-art REINFORCE recommender agent by introducing auxiliary tasks. We demonstrate that user response modeling as an instantiation of supervised auxiliary tasks is appealing for recommendation problems. We hypothesize the improvement comes from two aspects: 1) utilizing information from recommended items users have not interacted with; 2) introducing intermediate and easier tasks, i.e., predicting users' immediate response toward recommendations. Both guide the user and item representation learning, which ultimately benefits the policy learning to optimize for **long-term** user enjoyment of the platform. We share our experience in designing various aspects of the user response models, e.g., architecture and loss choices, treatment targeting and computational considerations. We also include a tool based on the gradient analysis to guide the development of the auxiliary task, e.g., architectural choices and the design of auxiliary losses. We demonstrate the effectiveness of our approach in both large-scale offline experiments on datasets with hundreds of millions of users and tens of millions of items, as well as experiments in a live system serving billions of users. We believe these are important steps in advancing the field of improving sample efficiency of reinforcement learning recommenders to scale to the user base and item corpus encountered in commercial recommendation platforms.

REFERENCES

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. *arXiv preprint arXiv:1705.10528* (2017).
[2] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. 2019. Striving for Simplicity in Off-policy Deep Reinforcement Learning. *arXiv preprint arXiv:1907.04543* (2019).
[3] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search*

and Data Mining. 456–464.
[4] Minmin Chen, Ramki Gummadi, Chris Harris, and Dale Schuurmans. 2019. Surrogate Objectives for Batch Policy Optimization in One-step Decision Making. In *Advances in Neural Information Processing Systems*. 8825–8835.
[5] 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).
[6] Corinna Cortes, Yishay Mansour, and Mehryar Mohri. 2010. Learning bounds for importance weighting. In *Advances in neural information processing systems*.
[7] 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.
[8] Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. 2018. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224* (2018).
[9] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679* (2015).
[10] Damien Ernst, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6, Apr (2005).
[11] Scott Fujimoto, David Meger, and Doina Precup. 2018. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900* (2018).
[12] 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.
[13] David Ha and Jürgen Schmidhuber. 2018. World models. *arXiv preprint arXiv:1803.10122* (2018).
[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. 2019. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*. 2555–2565.
[15] Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. 2017. Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475* (2017).
[16] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[17] Yujing Hu, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. 2018. Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 368–377.
[18] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. 2019. SlateQ: A tractable decomposition for reinforcement learning with recommendation sets. (2019).
[19] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. 2017. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*.
[20] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007* (2014).
[21] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. 2019. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374* (2019).
[22] Shivaram Kalyanakrishnan and Peter Stone. 2007. Batch reinforcement learning in a complex domain. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 94.
[23] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017).
[24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.
[25] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. 2019. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. *arXiv preprint arXiv:1906.00949* (2019).
[26] Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. In *Reinforcement learning*. Springer, 45–73.
[27] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643* (2020).
[28] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. 2018. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 37, 4-5 (2018), 421–436.
[29] Feng Liu, Ruiming Tang, Xutao Li, Weinan Zhang, Yunming Ye, Haokun Chen, Huifeng Guo, and Yuzhou Zhang. 2018. Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *arXiv preprint arXiv:1810.12027* (2018).
[30] Shikun Liu, Andrew Davison, and Edward Johns. 2019. Self-supervised generalisation with meta auxiliary learning. In *Advances in Neural Information Processing Systems*. 1679–1689.
[31] Taylor Mordan, Nicolas Thome, Gilles Henaff, and Matthieu Cord. 2018. Revisiting multi-task learning with rock: a deep residual auxiliary block for visual detection. In *Advances in Neural Information Processing Systems*. 1310–1322.
[32] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (2019), 54–71.
[33] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).
[34] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.
[35] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.
[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[37] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. 2018. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1134–1141.
[38] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005).
[39] 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.
[40] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.
[41] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. 2020. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136* (2020).
[42] Richard S Sutton, Andrew G Barto, et al. 1998. *Reinforcement learning: An introduction*. MIT press.
[43] 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.
[44] Adith Swaminathan and Thorsten Joachims. 2015. The self-normalized estimator for counterfactual learning. In *Advances in Neural Information Processing Systems*.
[45] Philip Thomas and Emma Brunskill. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *ICML*. 2139–2148.
[46] Shubham Toshniwal, Hao Tang, Liang Lu, and Karen Livescu. 2017. Multitask Learning with Low-Level Auxiliary Tasks for Encoder-Decoder Based Speech Recognition. *Proc. Interspeech 2017* (2017), 3532–3536.
[47] Trieu Trinh, Andrew Dai, Thang Luong, and Quoc Le. 2018. Learning Longer-term Dependencies in RNNs with Auxiliary Losses. In *International Conference on Machine Learning*. 4965–4974.
[48] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
[49] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
[50] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–38.
[51] Yuting Zhang, Kibok Lee, and Honglak Lee. 2016. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *International conference on machine learning*. 612–621.
[52] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. " Deep reinforcement learning for search, recommendation, and online advertising: a survey" by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. *ACM SIGWEB Newsletter* Spring (2019), 1–15.
[53] Xiangyu Zhao, Long Xia, Dawei Yin, and Jiliang Tang. 2019. Model-based reinforcement learning for whole-chain recommendations. *arXiv preprint arXiv:1902.03987* (2019).
[54] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2017. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209* (2017).
[55] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. (2018), 167–176.