Contents

1 Introduction

People confront the problem of vast space of options and choices now more than at any time in the past. We have enormous storages with all kinds of physical products and even more informational products like music, videos, knowledge, software, and services. Recommendation systems (RS) are used to facilitate the search in that space of goods and services.

Known common approaches to create RS are based on the information about the users, the products and their interactions, sometimes additional spatiotemporal information about the sequences of purchases is added to the model.

Various techniques, such as content-based collaborative filtering [1], matrix factorization methods [2, 3], logistic regression, factorization machines [4], contextual bandits [5] provide good performance in many cases.

Deep learning models are actively used in the construction of scalable RS. To grasp the landscape of different models you can consult this review [6].

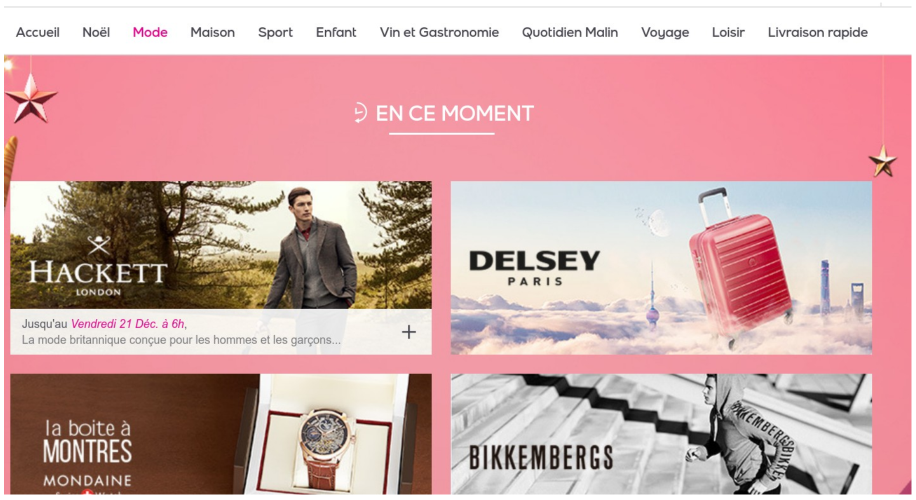
Usually, the recommendation process is viewed as a static, in other words, we train a model on some log history dataset and then serve it for some period of time without modifications. In order to update the model to account for users' and items' evolution in time, we need to retrain it. In that case, we do not model the influence of the interaction between user and RS and optimize only for a short-term reward (Compare: 'will a user like this product ?' vs 'will our RS provide agreeable user experience and increase overall usage of the service by a user ?').

The recent studies try to enter in the loop of recommendations to account for the effect of user-RS interaction and the evolution of user preferences through time. Treating the information of user-item interactions as a sequential decisionmaking problem is difficult but has several advantages.

By casting the problem of recommendations into Reinforcement Learning (RL) framework we hope to achieve several important benefits.

Firstly, we do not discard change in user's preference w.r.t. time. Secondly, we can optimize not only the immediate reward/feedback of the users to the item but also the longterm contributions that our recommendation policy can achieve.

Let's consider the following motivational example. We have an e-commerce company of event sales. They fulfill the supplier's needs to quickly sell excess inventory and now they cooperate with more than 6000 brands and make 54000 sales a year.



Jusqu'au *Vendredi 21 Déc. à 6h,*
La mode britannique conçue pour les hommes et les garçons...

Figure 1: Vente Privee web page

The set of available products change about 10% every week. This recurrent problem, also known as cold start, can be partially solved by using product and user embeddings to recommend similar products. We can train the model of deep matrix factorization with embeddings in the first place and then retrain it. It might not be very convenient but will solve that problem nevertheless.

Another concern is that in reality, our ultimate goal is to optimize life-long user content and not only a current reward, i.e. the preferences of users for each product at the moment.

We do not want to consider the task of recommendations as a static problem. The goal is to have a dynamic, online learning model. The sequential interaction between a service and a user can be modeled as a variant of a Markov Decision Process. Then by leveraging Reinforcement Learning, it might be possible to learn a good recommendation policy[7]. In other words that model can enter the circle of model-user interaction and integrate the understanding that the actions of the model dynamically affect user behavior.

At any time we work only with users immediate search and rank items accordingly. In that work, we tried to cast the problem of recommendation as a sequential decision problem (more precisely as a Markov Decision Process MDP) and solve it with Reinforcement Learning techniques.

2 Reinforcement learning

2.1 Notation

Many reinforcement learning problems can be defined as a policy search in a Markov decision process (MDP), defined as a tuple $(\mathcal{S}, \mathcal{A}, r, p, \gamma)$.

In the recommendation task the state space $\mathcal{S}$ and the action space $\mathcal{A}$ are assumed to be continuous as we use embeddings for items and users. State transition probability $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \inf)$ represent the probability density of the next state $s_{t+1} \in \mathcal{S}$ given the current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$. The environement outputs a reward $r : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ on each transition. In Figure 2 we can see more general view, where rewards can be viewed as a part of observation.[1] Policy of an agent is a function $\pi(a_t|s_t)$ that gives distribution over actions given a current state.

Markovian assumption $p(s_{t+1}|s_t, a_t, ..., s_1, a_1) = p(s_{t+1}|s_t, a_t)$ means that the next state depends only on the previous state and the chosen action.

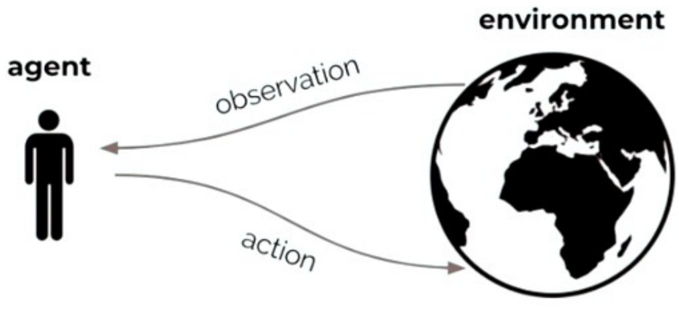


Figure 2: Agent interaction with environement

Agent proceeds as follows: starting at some initial state s_0 at each step agent chooses an action $a \sim \pi(a_t|s_t)$ and receives the reward r_t from the environement. The goal of the agent is to maximize the cumulative reward $G_\pi(s) = \sum_t \gamma^t r_t$

[1]DeepMind's RL course `https://www.youtube.com/playlist?list=PLqYmG7hTraZDNJre23vqCGIVpfZ_K2RZs`

Sometimes it is important to distinguish between observation and state. A state s is a complete description of the state of the world. There is no information about the world which is hidden from the state. An observation o is a partial description of a state, which may omit information. [8]. But in many articles authors will use them interchangeably. In this work we refer to the input/observation of the agent as state s.

One more particular moment will concern the representation of the state that the agent will consume. Without the loss of generality, we can say that an agent will take the state/observation provided by the environment by using some preprocessing of it (e.g. taking several previous states).

2.2 Basic concepts

In general environment transitions and the policy are stochastic.

$$P(\tau|\pi) = \rho_0 \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t) \tag{1}$$

And expected return can be computed as:

$$J(\pi) = \int_\tau P(\tau|\pi)R(\tau) = E_{\tau\sim\pi}[R(\tau)] \tag{2}$$

Our final problem is to find an optimal policy:

$$\pi^* = argmax_\pi J(\pi) \tag{3}$$

We can optimize parametric policy directly with policy optimization methods or we can express it with value or action-value functions.

Value function $V^\pi(s)$ give an expected return if you start in state s and follow the policy π

$$V^{\pi}(s) = E_{\tau\sim\pi}[R(\tau)|s_0 = s] \tag{4}$$

Action-Value function $Q^{\pi}(s,a)$ give an expected return if you start in state s, take arbitrary action a and follow the policy π aftewards

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a] \tag{5}$$

Optimal **Value function** and **Action-Value function** will be a fixed points of the Bellman equations:

$$V^{\pi}(s) = E_{a\sim\pi s'\sim P}[r(s,a) + \gamma V^{\pi}(s')] \tag{6}$$

$$Q^{\pi}(s,a) = E_{s'\sim P}[r(s,a) + \gamma E_{a'\sim\pi}[Q^{\pi}(s',a')]] \tag{7}$$

2.3 Classification of RL algorithms

This classification can be useful to better understand the variety of approaches and tasks.

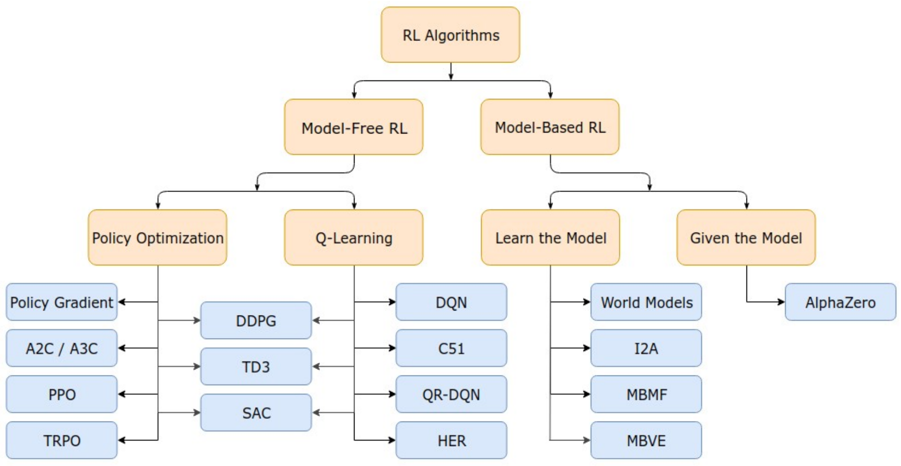


Figure 3: Classification of RL algorithms

It is not a complete classification but it gives an idea of the main features of the algorithms. Let's discuss some of them and explain which features are more suited for the task of recommendation.

2.3.1 Model Based vs Model Free

One of the most important branching points in an RL algorithm is the question of whether the agent has access to (or learns) a model of the environment. By a model of the environment, we mean a function which predicts state transitions and rewards. Algorithms which use a model are called model-based methods, and those that don't are called model-free.

In that work, we use model-free agents to interact with generated RS environments.

2.3.2 Q-learning vs Policy optimization

In Policy optimization methods we represent a policy explicitly as $\pi_\theta(a|s)$ and optimize it directly on some performance objective $J(\pi_\theta)$

In Q-Learning methods we learn an approximator of $Q_\theta(s, a)$ for optimal action-value function $Q^*_\theta(s, a)$. The corresponding policy is $a(s) = argmax_a Q_\theta(s, a)$

2.3.3 Off-policy vs On-policy

Off-policy can use the data collected at any point during training regardless of the behavior policy.

On-policy can only use a history obtained under the current policy with fixed parameters. After an update, we should create new trajectories.

In RS case the interactions with the users are costly and usually, there are plenty of historical logs obtained under previous versions of the production RS. Thus we tend to prefer Off-policy variants of the algorithms.

2.4 Deep reinforcement learning

For problems with huge state and action spaces, it is practical to use deep neural networks as a function approximators for policy, value or action-value functions. An introduction to deep reinforcement learning [9, 8]

2.5 Related works

We have defined notation, basic concepts and classification of methods of the reinforcement learning framework and now we will look at some examples of those spaces for specific recommendation tasks.

The amount of efforts invested in RL applications beyond ATARI games [10], Poker or Go is increasing. There was enough progress to stabilize the learning and improve sample efficiency for RL to become applicable in other areas like RS. In this section we will consider several recent works with promising approaches.

2.5.1 List-wise recommendations

To provide list-wise recommendations [11] it is possible to model a state space as a list of N last interacted items and an action space as a list of items. To make reward less sparse the authors proposed 3 different rewards skip/click/order. Each a_i is an embedding of product in some space or a feature vector because working with indices of items and users don't allow a policy to capture useful patterns efficiently. List-wise approach can provide users with diverse options.

More formally,

- **State space $\mathcal{S}$**: A state $s_t = \{s_t^1, \cdots, s_t^N\} \in \mathcal{S}$ is defined as the browsing history of a user, i.e., previous N items that a user browsed before time

t.

- **Action space $\mathcal{A}$**: An action $a_t = \{a_t^1, \cdots, a_t^K\} \in \mathcal{A}$

- **Reward $\mathcal{R}$**. The agent receives immediate reward $r(s_t, a_t)$ according to the user's feedback on each action in the list and a chosen aggregation rule, i.e. simple sum of rewards or weighted by position.

- **Transition probability $\mathcal{P}$**: Transition probability $p(s_{t+1}|s_t, a_t)$ defines the probability of state transition from s_t to s_{t+1} when RA takes action a_t. If user skips all the recommended items, then the next state $s_{t+1} = s_t$; while if the user clicks/orders part of items, then the next state s_{t+1} updates.

- **Discount factor γ**: $\gamma \in [0, 1]$ defines the discount factor when we measure the present value of future reward.

To handle large and dynamic state and action spaces they used Actor-Critic framework trained with DDPG procedure[12]. Given a state actor outputs parameters of a scoring function. The agent scores all items and selects the ones with the highest score without repetitions. Critic estimates the value of action for a given state.

For evaluation, they created Online Environment Simulator to define the reward for the state-action pairs that were not in the ground truth data. It is possible to build this simulator from a given sequential dataset.

In the testing phase model parameters are updated continuously and reset before each new session.

2.5.2 DRL with Explicit User-Item Interactions Modeling (Huawei)

Interesting statistics[13] that show the influence of the dynamic of the sequential interaction process between user and RS is the average rating of items rated after n consecutive positive (negative) ratings. Figure 4 shows significant dependence.

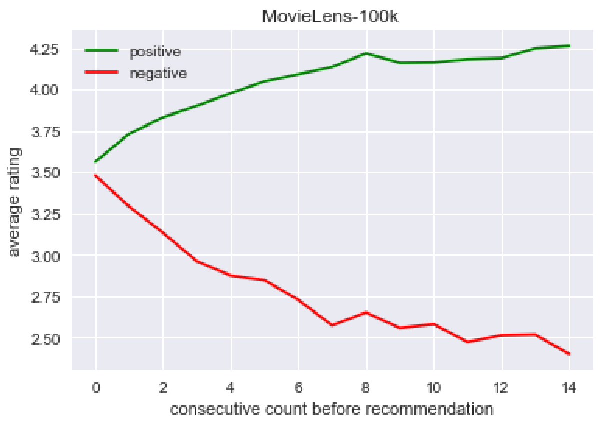


Figure 4: Average ranking depending from the number of precious consecutive rewards of the same type. We can observer strong influence of several bad rated interactions on the next rate.

The approach to define underlined MDP is similar to the paper with list-wise recommendations:

- **States** $\mathcal{S}$ - the representation of user's positive interaction history and her demographic information.
- **Actions** $\mathcal{A}$ - $a \in \mathbb{R}^k$ is a continuous parameter that will score each item $i_t \in \mathbb{R}^k$
- **Transitions** if interaction was positive it will be added to a user state
- **Reward** $R(s, a)$ click / not click / rating

There are 2 main differences in comparison to previous work:

- State representation module takes the N previous items and computes their interactions with the user and between items, thus modeling the user-item interactions implicitly.
- During the training procedure we compute the score of the items with weights given by actor, but the critic network receives the output of the policy network directly.

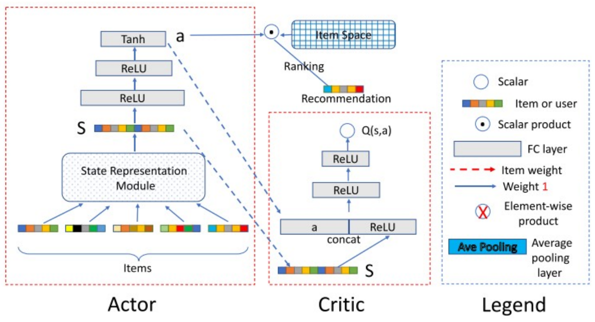


Figure 5: DRR framework

In this study, authors proposed a variant of evaluation RL recommender system using standard datasets such as MovieLens(100k), Yahoo! Music (R3), MovieLens(1M). However, their papers lack a precise description of the evaluation procedure. It is not clear how they split interactions by sessions and their shuffling procedure.

1. **Offline evaluation**: for a given session $\mathcal{S}_j$ agent only recommends items that appear in that session $\mathcal{I}(\mathcal{S}_j)$. After that agent observe reward $r_t = R(s_t, a_t)$ normalized to $[-1, 1]$, remove recommended item from the candidate set $\mathcal{I}(\mathcal{S}_j)$, update state and continue.
2. **Online simulator**: they trained Probabilistic Matrix Factorization (PMF) model on the whole dataset as the environment to be able to predict an item's feedback that the user never rates before.

Metrics used are Precision@k and NDCG@k for offline evaluation. And the total accumulated rewards for simulated online evaluation.

2.5.3 Top-K off-policy REINFORCE (Youtube)

In this work[7] researchers tried to come up with an approach to use huge quantities of logged implicit feedback that were collected under other recommendation policies. In order to avoid a bias in the estimation of the gradient for the policy gradient method they proposed Off-Policy correction.

It is possible to optimize the expected cumulative reward $max_\pi \mathbb{E}_{\tau\sim\pi} R(\tau)$, where $R(\tau) = \sum_{t=0}^{|\tau|} r(s_t, a_t)$ w.r.t to the policy parameters using gradient derived analytically with "log-trick"

$$\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)\nabla_\theta log\pi_\theta(\tau)] \approx \sum_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{|\tau|} R_t \nabla_\theta log\pi_\theta(a_t|s_t)\right] \tag{8}$$

That estimator is biased if the historical policy β from which we take trajectories τ is not equal to the policy of an agent π_θ. And this is the case of RS, because we have a lot of logs from previous versions of other RS. Importance Sampling (IS) is a standard way of correcting that bias.

$$\sum_{\tau\sim\beta}\left[\sum_{t=0}^{|r|} \frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)} R_t \nabla_\theta log\pi_\theta(a_t|s_t)\right] \tag{9}$$

The drawback of that approach is that it will increase variance. There are several options to control it : Weight Capping[7], Normalized Importance Sampling[7], Trusted Region Policy Optimization (TRPO).

To make list-wise recommendations they assumed that the expected reward of a set of non-repetitive items equals to the sum of rewards of each item. They generated the action by independently sampling according policy π_θ

$$\alpha_\theta(a|s) = 1 - (1 - \pi_\theta(a|s))^K \tag{10}$$

is the probability for item a to appear in a final recommendation of size K. We change π_θ to α_θ.

Thus, we can rewrite the gradient:

$$\sum_{\tau \sim \beta} \left[\sum_{t=0}^{|r|} \frac{\pi_\theta(a_t|s_t)}{\beta(a_t|s_t)} \frac{\partial \alpha(a_t|s_t)}{\partial \pi(a_t|s_t)} R_t \nabla_\theta log \pi_\theta(a_t|s_t) \right] \tag{11}$$

State is a vector $S_t \in \mathbb{R}^n$, actions at each time step is embedded to $u_a \in \mathbb{R}^m$. To model a state transition $S_{t+1} = f(S_t, a_t)$ they used RNN (CFN). RNN will output next state S_{t+1}, after processing some previous actions, then we can compute the probability of actions using softmax over scores that is an inner product $s^T v_a$ where $v_a \in \mathbb{R}^n$ is another embedding of the item.

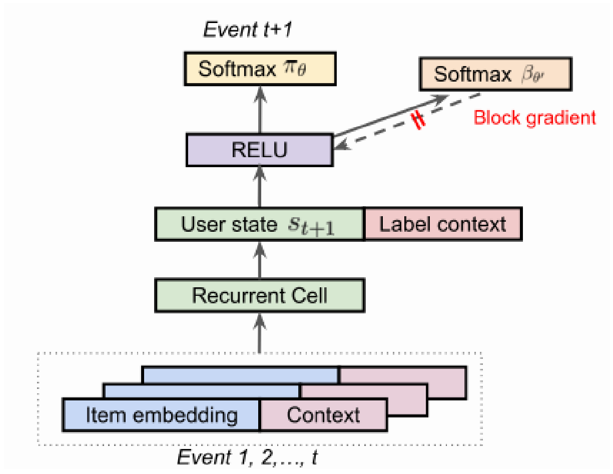


Figure 6: Policy network

To estimate $\beta(a|s)$ we can use the same architecture of a policy network. But that softmax will be over a different set of possible items from log.

It is computationally demanding to compute full softmax over all items available for recommendations at YouTube, so the researchers avoid that by using fast nearest neighbor algorithm[14] to extract several items with the most probability mass according to score and run small softmax on them.

Production experiments have shown small but statistically significant lift in ViewTime 0.07%. The good feature was that the agent policy diverged from the behavior policy and gave more probability to rare items without loss in observed metrics.

3 Agent models

In this section, we will describe our agents used for the experiments.

3.1 DQN

First, we adapt basic Deep Q-Network (DQN)[15] to make recommendations. The difference from ATARI settings is that we have a set of available items that changes potentially each step. And we also want to recommend several items from that list.

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
Initialise sequence $s_1$, $A_1 = a_1, ..., a_{N_1}$
for $t = 1, T$ do
    With probability $\epsilon$ select a random action $a_t = \{i_1, ..., i_K\}$
    otherwise select top $K$ items by $argmax_i Q_\theta(s_t, a_i)$
    Execute action $a_t$ in emulator and observe reward $r_t$, state $s_{t+1}$ and possible items $A_{t+1}$
    Store transition $(s_t, A_t(a_t), r_t, s_{t+1}, A_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions from $\mathcal{D}$ :
        $(s_j, A_j(a_j), r_j, s_{j+1}, A_{j+1})$
    Set $y_j = r_j + \gamma \sum_{k=1}^{K} \max_{a'_k} Q_{\theta_{target}}(s_{j+1}, a'_k)$
    Perform a gradient descent step on $\left(y_j - \sum_{k=1}^{K} Q_\theta(s_j, a_{j_k})\right)^2$
    update $\theta_{target} = \theta$ each $M$ steps
end for
```

At each step, based on the current state S_t, the agent will score every item from possible items set $\mathcal{A}_t$ and select K items with maximal $Q(S_t, a_i)$. Then environment will give reward R_{t+1}, the next state S_{t+1} and possible items for the next step $\mathcal{A}_{t+1}$. Tuple $(S_t, a_t, R_{t+1}, S_{t+1}, \mathcal{A}_{t+1})$ is saved to the replay buffer.

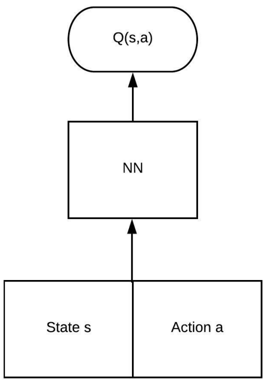


Figure 7: Action-Value function for DQN agent. We cannot make outputs for each action, because the action space is changing and it might be big

Then network weights are optimized by SGD to minimize the loss:

$$[R_{t+1} + \gamma \sum_{a'} Q_{\theta_{target}}(S_{t+1}, a') - \sum_{a \in a_t} Q_\theta(S_t, a)]^2$$

Note that to use list-wise recommendations we use the sum of Q-values for each action in the recommendation. And we also take K max items by Q-value at next state S_{t+1}

We used 2 FC layers with 200 and 100 hidden units and relu activation. Learning rate is $1e-4$ and $\gamma = 0.9$. Optimized by Adam.

Action is chosen with ϵ-greedy strategy, that gives random items with probability ϵ.

The main drawback of that approach is that we need to score every item using one feedforward pass hrough neural network. It is very computationally expensive if we have a lot of items to score. So we need another approach.

3.2 DDPG

Deep Deterministic Policy Gradient is an off-policy algorithm that scales well with huge action spaces. There are two parts: actor that model deterministic policy $a(s)$ and critic that approximate action-value function $Q(s, a)$.

Actor network models state-specific scoring function $f_{\theta^{\pi}} : s_t \rightarrow w_t$ Then for each position i in the recommendation vector we take the item that maximize $score_i = w_t^k i_i^T)$ without repetition.

Critic takes state S_t and action a_t and computes $Q(S_t, a_t)$.

Optimal policy should satisfy:

$$a^*(s) = arg \max_a Q^*(s, a) \tag{12}$$

Using that fact we can avoid computing max over action-values of each item as in DQN, instead we will just use $Q(s, a(s))$.

Critic minimizes mean-squared Bellman error (the difference between bootstrapped action-value and current action-value)

$$\mathcal{L} = \mathbb{E}\left[(Q(s, a) - (r + \gamma * Q_{target}(s', a_{target}(s'))^2\right] \tag{13}$$

Actor's weights are also updated with gradient descent to maximaze:

$$\max_{\theta} \mathbb{E}[Q(s, a_{\theta}(s))] \tag{14}$$

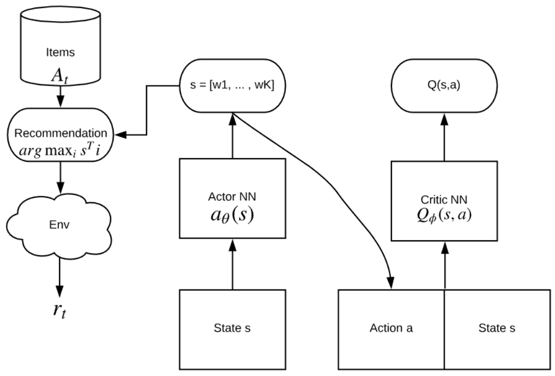


Figure 8: DDPG

Algorithm 2 Deep Deterministic Policy Gradient

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
Set target parameters $\theta_{target} \leftarrow \theta$, $\phi_{target} \leftarrow \phi$
Initialise sequence $s_1$, $A_1 = a_1, ..., a_{N_1}$
for $t = 1, T$ do
    Observe state $s$
    Actor returns scoring function $a_t = a_\theta(s_t)$, select items as $argmax_i a_t^T i$
    Observe reward for chosen items $r_t$, state $s_{t+1}$ and possible items $A_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1}, d)$ in $\mathcal{D}$
    Sample random minibatch $B$ of transitions from $\mathcal{D}$ :
        $$(s_j, a_j, r_j, s_{j+1}, d_j)$$
    Set target $y_j = r_j + \gamma(1 - d_j)Q_{\phi_{target}}(s_{j+1}, a_{target}(s_{j+1}))$
    Update Q-function by one step of gradient DESCENT:
        $$\nabla_\phi \frac{1}{|B|} \sum_j (y_j - Q_\phi(s_j, a_j))^2$$
    Update policy by one step of gradient ASCENT:
        $$\nabla_\theta \frac{1}{|B|} \sum_j Q_\phi(s_j, a_\theta(s_j))$$
    update target networks
        $$\theta_{target} = \theta$$
        $$\phi_{target} = \phi$$
each $M$ steps
end for
```

We used 2 FC layers with 64 and 32 hidden units and tanh activation for actor and (32, 32) hidden units for the critic. Learning rate is $1e-4$ and $\gamma = 0.9$. Optimized by Adam. Batch size is 64. Replay buffer size is 10000.

Modifications: prioritized experience replay [16]

3.3 Baselines

- **Random Agent** - sample randomly K items from A_t
- **Deep Matrix Factorization** - concatenate user and item embeddings and put in 2 fully connected layers.
- **Popularity Agent** - recommend items with highest mean rank so far, or highest number of positive ratings
- **LinUCB** - linear contextual bandit with disoint features [17]
- **HLinUCB** - linear contextual bandit with shared features [17]
- **PPO2** - Proximal Policy Optimization from stable baselines, that combines the ideas of actor-critic and constrained update of TRPO.

4 Environment

We mentioned earlier the sample inefficiency problem. Other problem might be that there are not many datasets for recommendations with proper time information. If we have a proprietary data we can proceed by constructing a simulator of rewards as in [18]. And then train agent with that environment.

Another approach is to build some model by ourselves with some assumptions on distributions of user preferences. For example, [19] proposed RecoGym environment to model a process of web-ads clicking.

To construct such environment it is better to use OpenAI gym [20] library that proposes a convenient simple interface. Then it can be easily shared, thus facilitating reproducibility. Two main methods reset and step is introduced in Fig. 11. The episode is an interaction with one user, it starts by reset method. The episode is considered as finished when the step method return $done = True$.

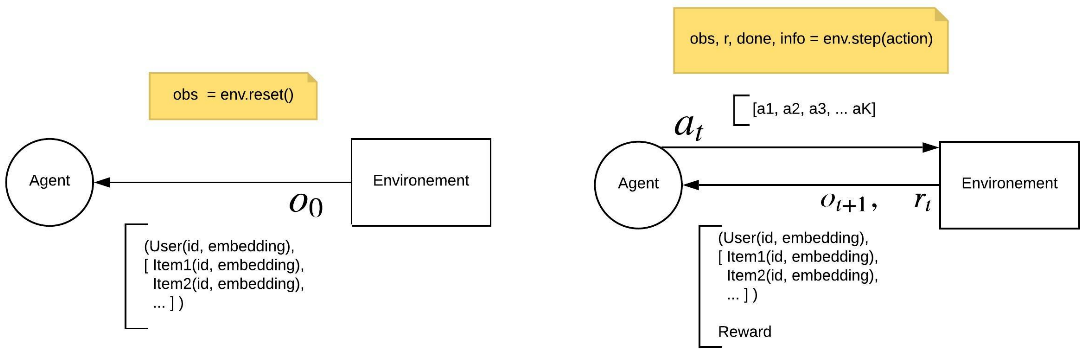

obs = env.reset()
Agent
Environement
o_0
(User(id, embedding), [Item1(id, embedding), Item2(id, embedding), ...])
obs, r, done, info = env.step(action)
a_t
[a1, a2, a3, ... aK]
Agent
Environement
o_{t+1}, r_t
(User(id, embedding), [Item1(id, embedding), Item2(id, embedding), ...])
Reward

Figure 9: Interaction loop with environment

4.1 MovieLens offline

One straightforward approach to evaluation is an offline evaluation of the existing dataset.

	ml-100k	ml-1m
users	943	6040
items	1682	3952
ratings	100000	1000209

Table 1: Dataset statistics

To perform offline evaluation we will ask the algorithm to rerank known items from user sessions. The movies have rantings from 1 to 5. We will consider movies with ratings 4 and 5 as relevant to compute our metrics.

We can split interactions into sessions by time or just by a number of interactions.

Algorithm 3 Offline Evaluation Algorithm

```
$n$ state window size
$R$ reward function
Observe initial state $s_0$
Observe set $\mathcal{I}$ from log
**for** t = 1, T **do**
    Observe current state $s_t = \{i_1, ..., i_n\}$
    Execute action $a_t = \pi_\theta(s_t)$
    Choose $i_t$ with highest score
    Get reward $r_t = R(s_t, a_t)$ from log
    Update state $s_{t+1} = f(H_{t+1})$
    Remove $i_t$ from $\mathcal{I}$
**end for**
```

We compute the following metrics: precision@k, NDCG@k.

$$\text{precision} = \frac{\text{\# of receommended in first k in } \mathcal{I}}{\text{\# of recommended}} \tag{15}$$

$$\text{DCG@k} = \sum_{i=1}^{k} \frac{r_i}{log(1+i)} \quad \text{nDCG@k} = \frac{DCG@k}{IDCG@k} \tag{16}$$

where $IDCG@k$ is Ideal Discounted Cumulative gain, where recommended items sorted by their relevance

At each time step we recommend $K = 1$ items, metrics @k are computed for all recommendations during one session. sXX gives the size of the session in items.

	ml-100k s20		ml-100k s30	
	NDCG@10	Precision@10	NDCG@10	Precision@10
Random	0.7	0.15	0.68	0.18
Popularity	0.85	**0.21**	0.82	0.22
SVD	0.82	0.18	0.78	0.18
LinUCB	**0.86**	0.20	**0.83**	**0.23**
HLinUCB	0.83	0.19	0.82	0.21
DDPG	0.84	0.20	0.81	0.22
PPO2	0.82	0.18	0.81	0.21

Table 2: Metrics for different session size for MovieLens100k

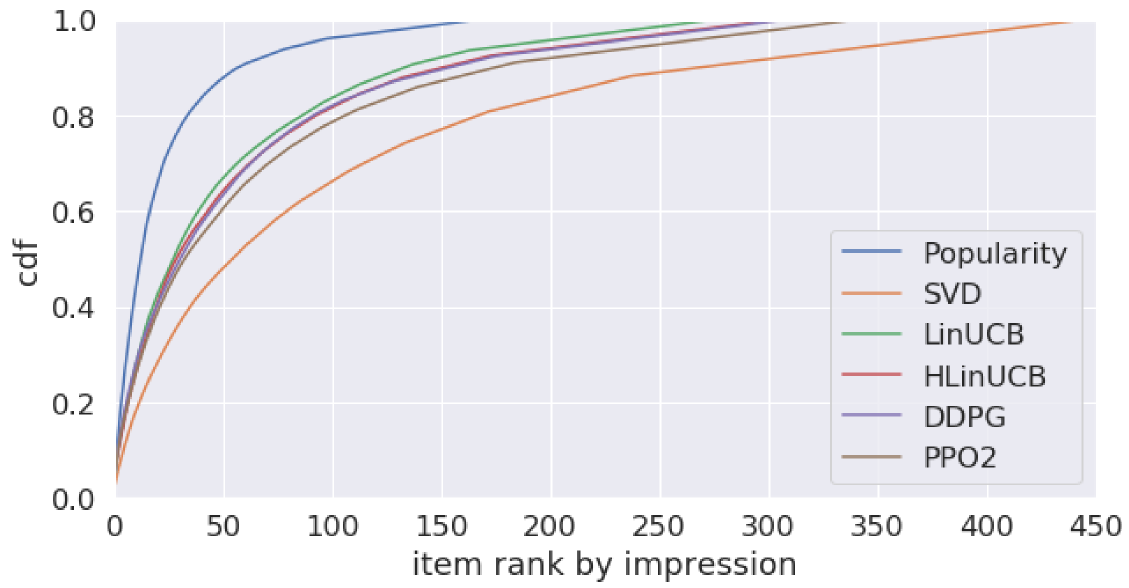


Figure 10: Distribution of impressions over items: lower curve means that algorithm served more different items

As we can see results of RL algorithms are quite comparable with LinUCB but provides slightly better variety as their CDF curve are lower.

	ml-1m s20		ml-1m s30	
	NDCG@10	Precision@10	NDCG@10	Precision@10
Random	0.76	0.31	0.77	0.32
Popularity	0.83	0.44	0.84	0.44
SVD	0.78	0.33	0.77	0.32
LinUCB	**0.85**	**0.44**	**0.86**	**0.45**
DDPG	0.82	0.40	0.83	0.4
PPO2	0.79	0.35	0.84	0.42

Table 3: Metrics for different session size for MovieLens 1M

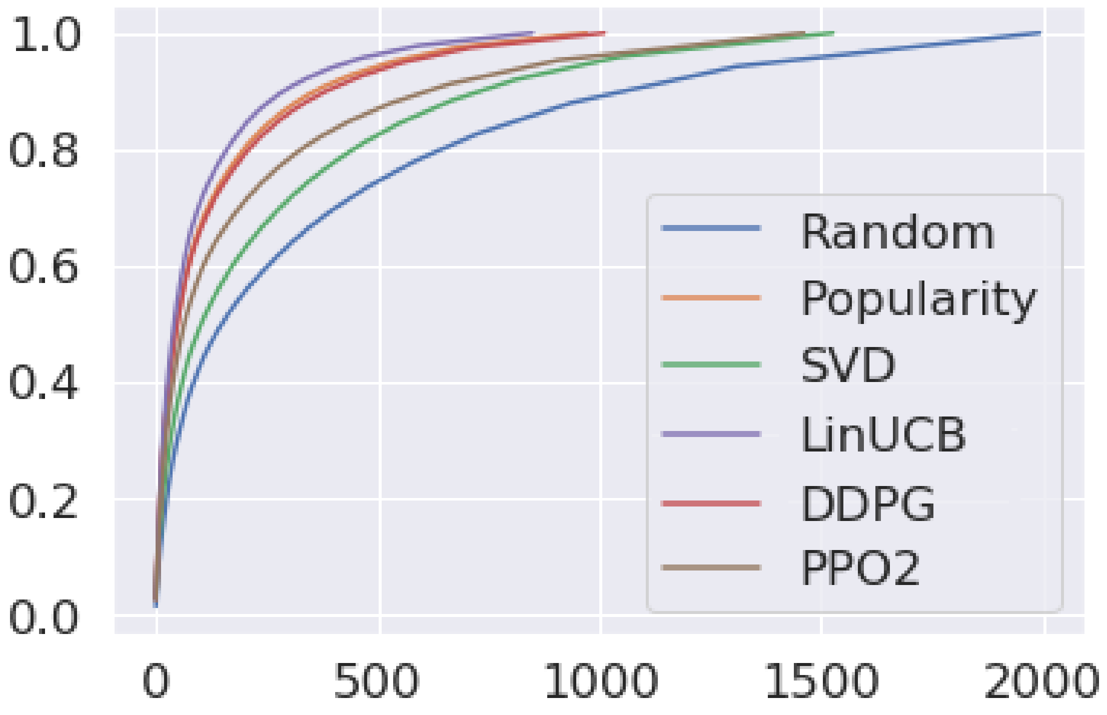


Figure 11: Distribution of impressions over items for MovieLens 1M: lower curve means that algorithm served more different items

4.2 MovieLens

The main problem of offline evaluation is that we can't obtain the rewards for the actions that we didn't take in out logged policies. In order to obtain some model of rewards for any pair of user-item in the dataset, we create the embeddings of users and items by performing Probabilistic Matrix Factorization (PMF) over MovieLens-100k dataset. The recommendation process proceeds as follows: environment gives a user and a set of available for recommendation items to the agent. Then agent chooses K elements and returns their indexes. The environment returns the agent reward and new observation.

The reward can be a score from 1 to 5 predicted by PMF (i.e. dot product between user and item embeddings) or it can be normalized to [-1,1].

Algorithm 4 Online Evaluation Algorithm

```
Train PMF item/user vectors
Update parameters continuously
restore the parameters from train stage before each evaluation session
```

As we give all not interacted items to the agent, it takes a considerable amount of time to score all items, so we took a smaller environment with 1000 user and 1000 items. (The embeddings were constructed using the whole MovieLens-100k dataset).

At each time step agent receive a user representation as 5 last interacted items and all items that the user hasn't seen. Then the agent recommends 1 item and receives the reward and next user/item observation.

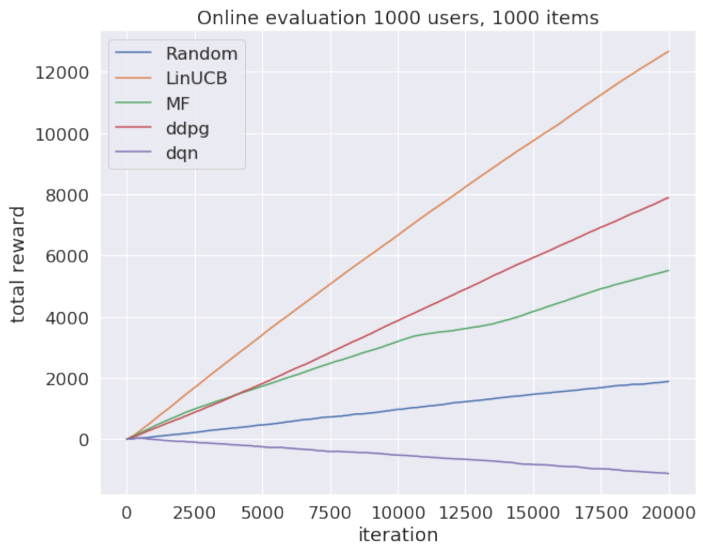


Figure 12: Total accumulated reward for MovieLens-v2 rewards [-1, 1]

Even from quite a few steps DDPG could capture some dependencies better than Deep matrix factorization (MF). But basic DQN failed in such task.

5 Discussion

With current progress in the understanding of Reinforcement learning, it is possible to obtain algorithms stable enough to use in recommender systems. RL algorithms performed a bit worse on precision@k and NDCG@k on a proposed offline benchmark, but they proposed better exploration than other baselines.

There appear many articles from Google[7], Facebook[21], Huawei[13], Adobe[22], Alibaba[23, 24] and JD.com[25] on that topic in the recent 2 years. Authors proposed different representations of underlying MDP and the algorithms to find optimal policies. Although in several works only marginally improvements were observed in production A/B testing, the authors noticed that RL approaches serves more varied items. If the recommender system serves more different items without loss in Click-Through-Rate (CTR) or view-time, that means in the future it will have more information about the items and the user preferences, which will lead to further improvements. But that effect is more difficult to measure in short A/B testing phase.

Reinforcement learning is an especially promissing research topic in the scientific community. There are still a lot of theoretical problems concerning stability and convergence.

6 Conclusion

In this project, we have studied the applications of reinforcement learning algorithms to the task of recommendations.

The main contributions of the project are 3 folds:

1. Discussed several popular approaches for casting recommendations in the MDP framework for reinforcement learning

2. Built several types of recommendation environments/benchmarks

3. Experiments with DQN and DDPG agents

That research direction is very promising and vast. Extensive ablation studies and comparison between different MDP models could be fruitful directions of future research.

Bibliography

1. *Linden G.*, *Smith B.*, *York J.* Amazon.com Recommendations: Item-to-Item Collaborative Filtering // IEEE Internet Computing. — 2003. — Jan. — Vol. 7. — P. 76–80. — ISSN 1089-7801. — DOI: 10.1109/MIC.2003.1167344. — URL: doi.ieeecomputersociety.org/10.1109/MIC.2003.1167344.

2. *Xue H.-J.* [et al.]. Deep Matrix Factorization Models for Recommender Systems. // IJCAI. — 2017. — P. 3203–3209.

3. *Bonner S.*, *Vasile F.* Causal Embeddings for Recommendation // Proceedings of the 12th ACM Conference on Recommender Systems. — Vancouver, British Columbia, Canada : ACM, 2018. — P. 104–112. — (RecSys '18). — ISBN 978-1-4503-5901-6. — DOI: 10.1145/3240323.3240360. — URL: http://doi.acm.org/10.1145/3240323.3240360.

4. *McMahan H. B.* [et al.]. Ad Click Prediction: A View from the Trenches // Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. — Chicago, Illinois, USA : ACM, 2013. — P. 1222–1230. — (KDD '13). — ISBN 978-1-4503-2174-7. — DOI: 10.1145/2487575.2488200. — URL: http://doi.acm.org/10.1145/2487575.2488200.

5. *Li L.* [et al.]. A Contextual-bandit Approach to Personalized News Article Recommendation // Proceedings of the 19th International Conference on World Wide Web. — Raleigh, North Carolina, USA : ACM, 2010. — P. 661–670. — (WWW '10). — ISBN 978-1-60558-799-8. — DOI: 10.1145/1772690.1772758. — URL: http://doi.acm.org/10.1145/1772690.1772758.

6. *Zhang S.* [et al.]. Deep Learning based Recommender System: A Survey and New Perspectives. — 2017. — arXiv: 1707.07435. — URL: https://arxiv.org/abs/1707.07435.

7. *Chen M.* [et al.]. Top-K Off-Policy Correction for a REINFORCE Recommender System. — 2018. — arXiv: 1812.02353. — URL: http://arxiv.org/abs/1812.02353.

8. *Achiam J.* OpenAI Spinning Up in Deep RL! — 2019. — URL: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html.

9. *François-Lavet V.* [et al.]. An Introduction to Deep Reinforcement Learning // CoRR. — 2018. — Vol. abs/1811.12560. — arXiv: 1811.12560. — URL: http://arxiv.org/abs/1811.12560.

10. *Mnih V.* [et al.]. Playing Atari with Deep Reinforcement Learning // CoRR. — 2013. — Vol. abs/1312.5602. — arXiv: 1312.5602. — URL: http://arxiv.org/abs/1312.5602.

11. *Zhao X.* [et al.]. Deep Reinforcement Learning for List-wise Recommendations. — 2017. — arXiv: 1801.00209. — URL: http://arxiv.org/abs/1801.00209.

12. *Lillicrap T. P.* [et al.]. Continuous control with deep reinforcement learning. — 2015. — arXiv: 1509.02971 [cs.LG].

13. *Liu F.* [et al.]. Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling. — 2018. — arXiv: 1810.12027. — URL: http://arxiv.org/abs/1810.12027.

14. *Guo R.* [et al.]. Quantization based Fast Inner Product Search. — 2015. — arXiv: 1509.01469. — URL: http://arxiv.org/abs/1509.01469.

15. *Mnih V.* [et al.]. Playing Atari with Deep Reinforcement Learning. — 2013. — arXiv: 1312.5602. — URL: http://arxiv.org/abs/1312.5602.

16. *Schaul T.* [et al.]. Prioritized experience replay // arXiv preprint arXiv:1511.059
2015.

17. *Li L.* [et al.]. A Contextual-Bandit Approach to Personalized News Article Recommendation. — ISBN 9781605587998. — URL: http://rob.schapire.net/papers/www10.pdf.

18. *Zhao X.* [et al.]. Recommendations with Negative Feedback via Pairwise Deep Reinforcement Learning. — 2018. — DOI: 10.1145/3219819.3219886. — arXiv: 1802.06501. — URL: http://arxiv.org/abs/1802.06501http://dx.doi.org/10.1145/3219819.3219886.

19. *Rohde D.* [et al.]. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. — 2018. — DOI: 10.1111/j.1540-8159.2011.03285.x. — arXiv: 1808.00720. — URL: http://arxiv.org/abs/1808.00720.

20. *Brockman G.* [et al.]. OpenAI Gym. — 2016. — arXiv: 1606.01540. — URL: http://arxiv.org/abs/1606.01540.

21. *Gauci J.* [et al.]. Horizon: Facebook's Open Source Applied Reinforcement Learning Platform. — 2018. — arXiv: 1811.00260. — URL: http://arxiv.org/abs/1811.00260.

22. *Theocharous G.*, *Thomas P. S.*, *Ghavamzadeh M.* Personalized ad recommendation systems for life-time value optimization with guarantees // IJCAI Int. Jt. Conf. Artif. Intell. 2015-Janua. — 2015. — P. 1806–1812. — ISBN 9781577357384. — DOI: 10.1145/2740908.2741998. — URL: https://www.ijcai.org/Proceedings/15/Papers/257.pdf.

23. *Liu Y.* [et al.]. Diversity-Promoting Deep Reinforcement Learning for Interactive Recommendation. — 2019. — arXiv: `1903.07826`. — URL: `http://arxiv.org/abs/1903.07826`.

24. *Hu Y.* [et al.]. Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application. — 2018. — arXiv: `1803.00710`. — URL: `http://arxiv.org/abs/1803.00710`.

25. *Zhao X.* [et al.]. Model-Based Reinforcement Learning for Whole-Chain Recommendations. — 2019. — arXiv: `1902.03987`. — URL: `http://arxiv.org/abs/1902.03987`.