Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling

Reinforcement Learning final project

Annalisa Paladino, Luca Pernice, Emanuele Ruoppolo

Introduction

Aim: create a dynamic recommendation system

Symbiotic relationship between **users** and the **recommendation** model

The model provides recommendations to users, helping them come to a **decision**, and users provide feedback to the model, helping the model make **better recommendations**

Non-RL Recommendation Systems

- ❑ Content-based collaborative filtering
- ❑ Matrix factorization based methods
- ❑ Logistic regression
- ❑ Factorization machines and its variants
- ❑ Deep learning models

Limitations:

1. Consider the recommendation procedure as a **static** process
2. Trained by maximizing the *immediate* rewards of recommendations, but ignores the ***long-*** **term** contributions that the items can make

RL Recommendation Systems

- **Model-based** (eg. POMDP and Q-learning): inapplicable when the number of candidate items is large, because a time-consuming dynamic programming step is required to update the model

- **Model-free**

Value-based: very inefficient if the action space is too large

Policy-based: the interactions between users and items is not explicitly modeled

Translation into a RL framework

Agent: recommendation system

Environment: users

State: representation of a user and his most recent positively rated items

Actions: continuous vector weighting items in a candidate set

Reward: user’s rating of an item

Discount factor: measure the present value of long-term rewards

Recommender-User interactions in MDP

At timestep t, the **state** can be defined as

$$s_t = f(H_t)$$

state representation module

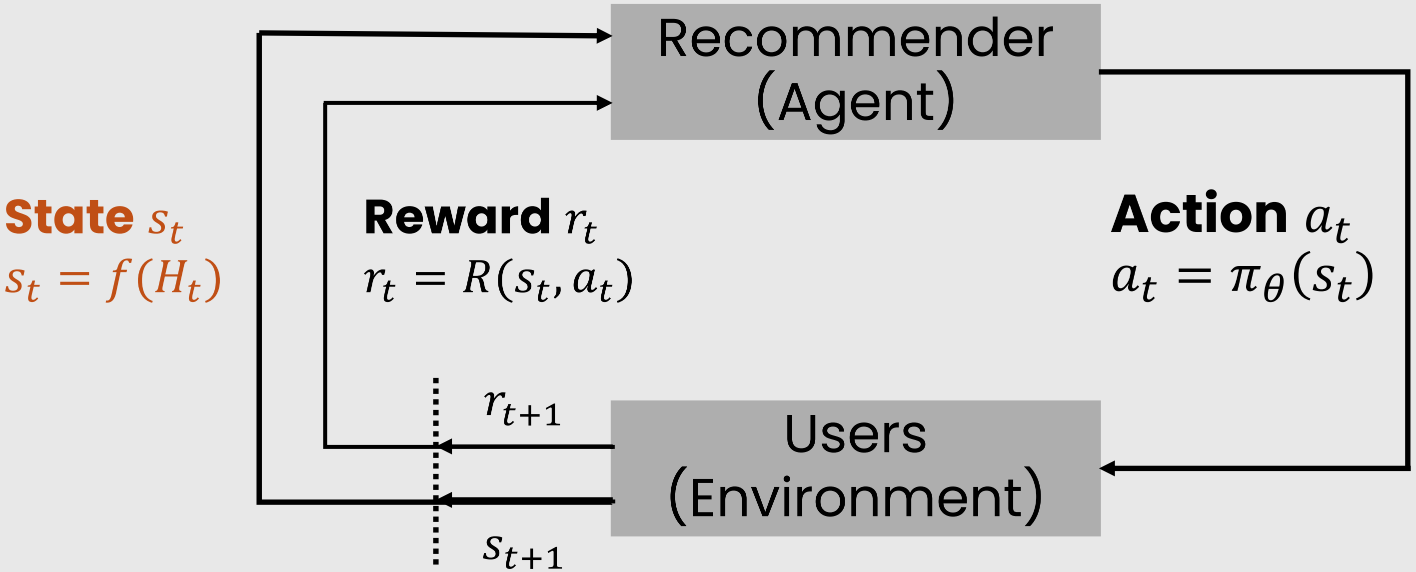
$H_t = \{i_1, \dots, i_n\}$ are the embeddings of the latest positive interaction history

i_t is a k-dimensional vector

Why?

1. A superior recommender system should cater to the users' taste, what items the users like

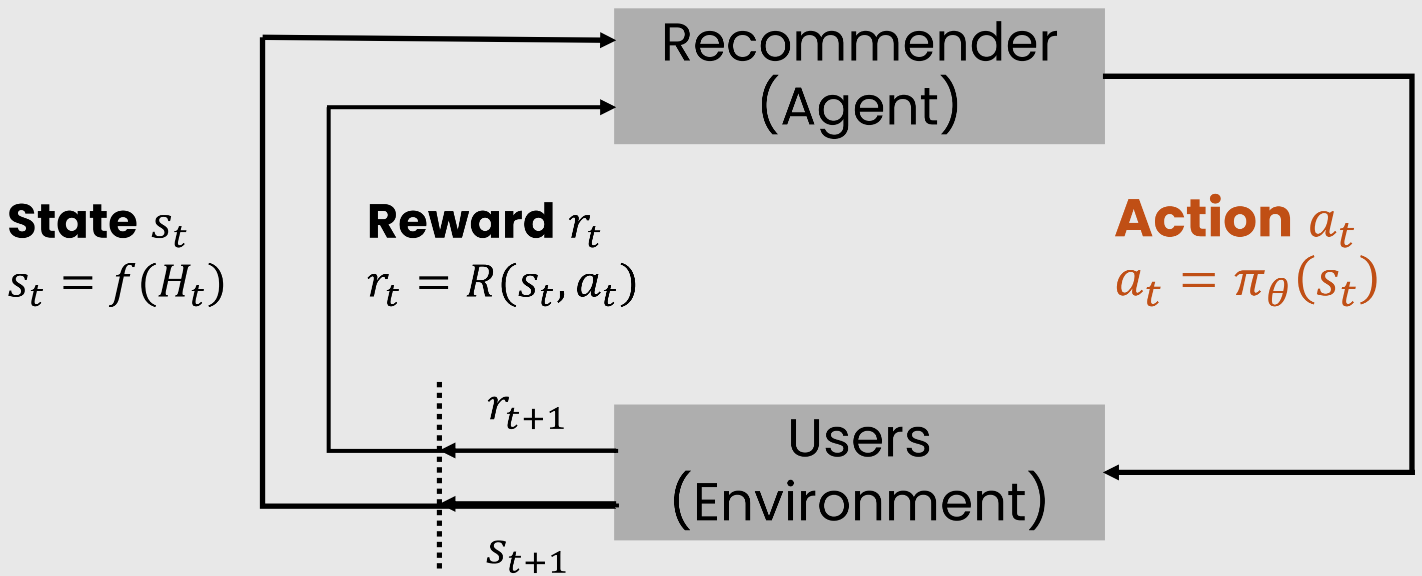
2. The latest records represent the users' recent interests more precisely.



Recommender – User interactions in MDP

An **action** is a continuous parameter vector that is used to determine the **ranking scores** of all the items

All the candidate items are ranked according to the computed scores and Top-N items are recommended to the user

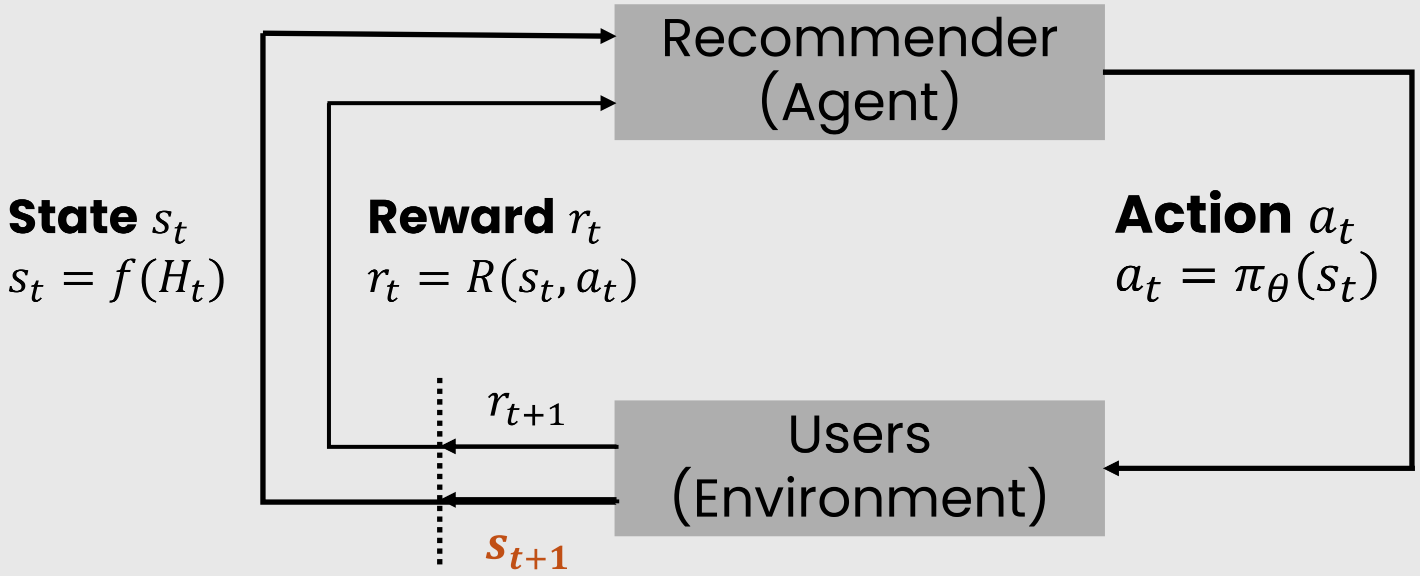


Recommender–User interactions in MDP

When the recommender agent recommends an item i_t, if the user provides *positive feedback*, then in the **next timestep**, the state is updated to

$$s_{t+1} = f(H_{t+1})$$

$$H_{t+1} = \{i_2, \dots, i_n, i_t\}$$

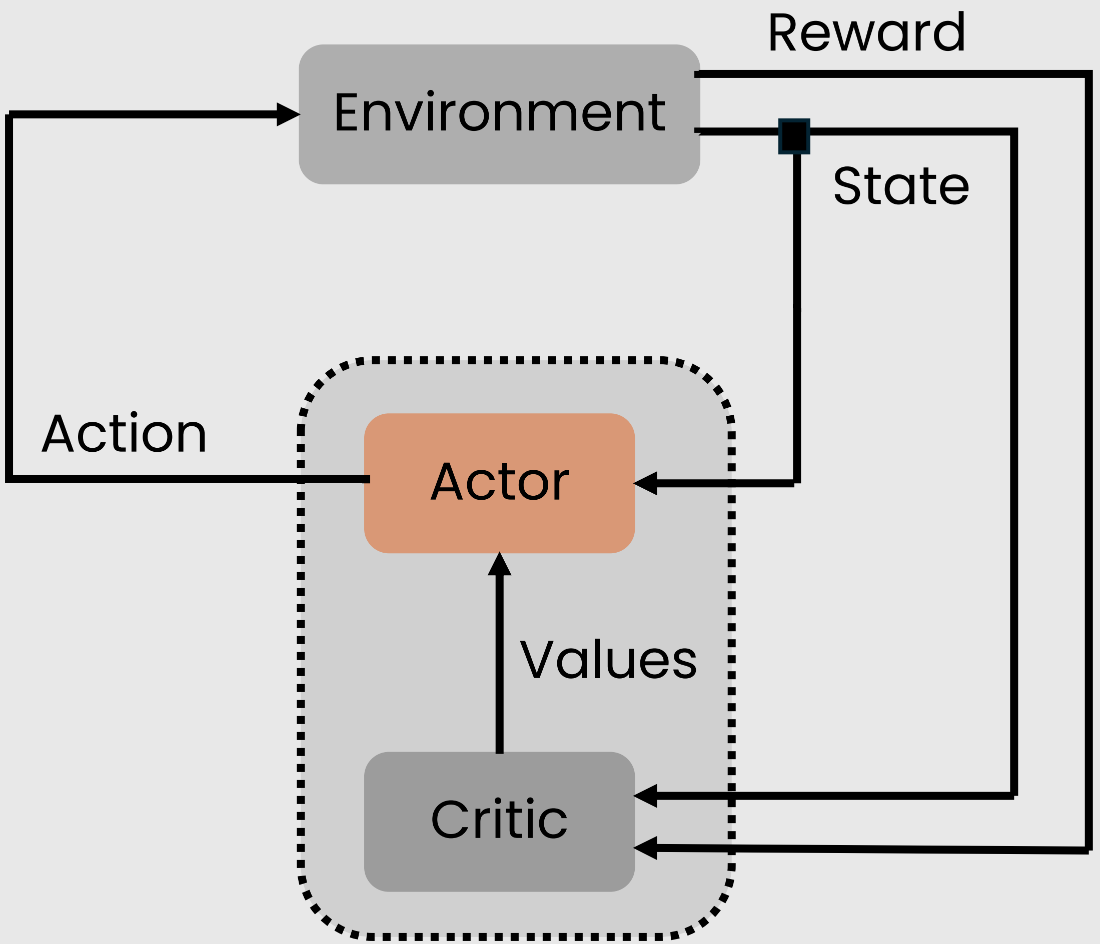


BACKGROUND

Actor – **Critic**

Actor

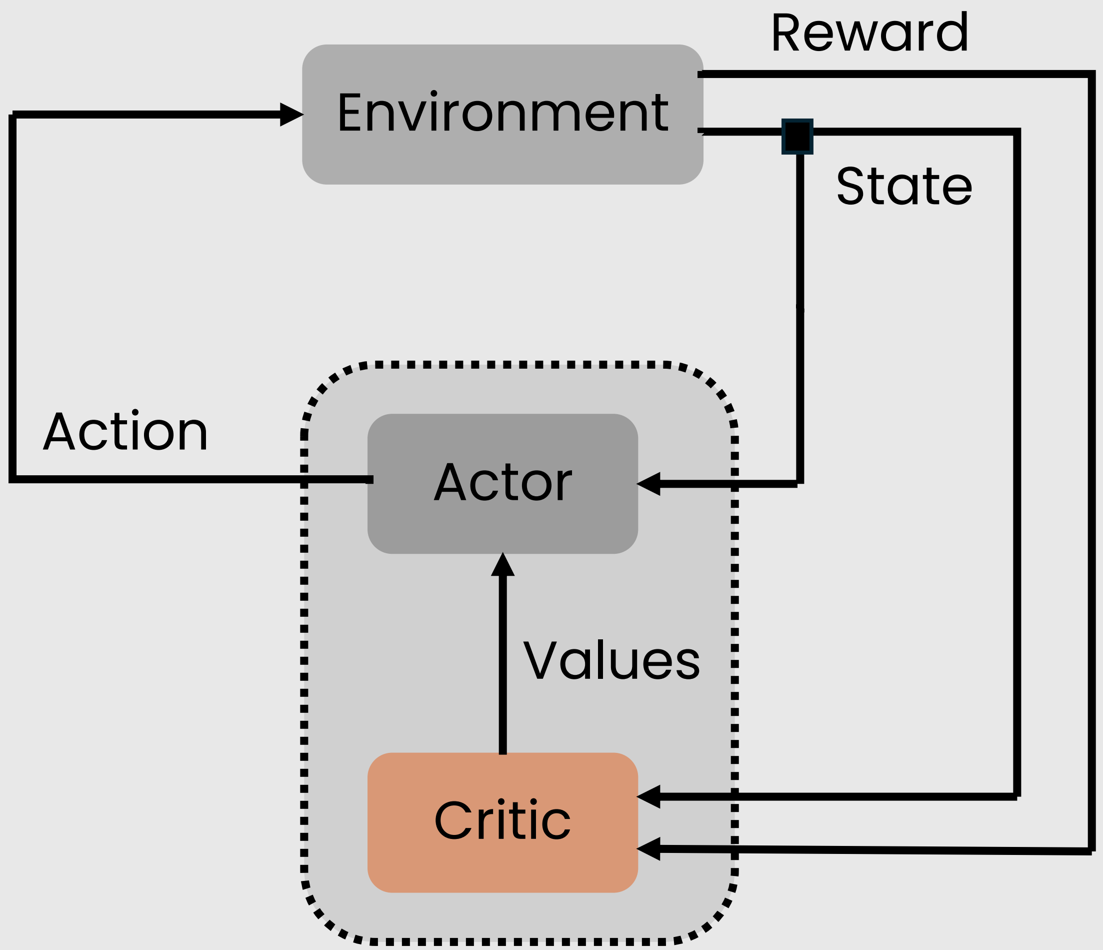
An *agent* (the "actor") makes decisions by selecting actions based on the current **policy**. It explores the action space to maximize expected cumulative rewards. By continuously refining the policy, the actor adapts to the dynamic nature of the environment



Actor – Critic

Critic

A *value function* (the “critic”) evaluates the actions taken by the actor. It estimates the **value** or quality of these actions by providing feedback on their performance. The critic’s role is to guide the actor towards actions that lead to higher expected returns, contributing to the overall improvement of the learning process



Deep Deterministic Policy Gradient (DDPG)

DDPG is an **off-policy actor-critic** based algorithm

Key components:

- **Critic Network** $Q(s, a|\theta^Q)$: it estimates the action-value function, which predicts the expected return of a given state-action pair
- **Actor Network** $\mu(s|\theta^\mu)$: it outputs the action to be taken given a state
- **Target Networks** Q' and μ': these are copies of the critic and actor networks used for stable updates
- **Replay Buffer** R: buffer to store experience tuples (s_t, a_t, r_t, s_{t+1})

DDPG Algorithm

Algorithm DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Implementation

Suppose we're able to store a *user-item interaction matrix* ***R***.

Some cells are empty because there are many items a user do not review (R is a **sparse matrix**). But we could fill the empty cells with predictions...

Implementation

PROBLEMS:

- Matrix R is too large
- How do we predict ratings

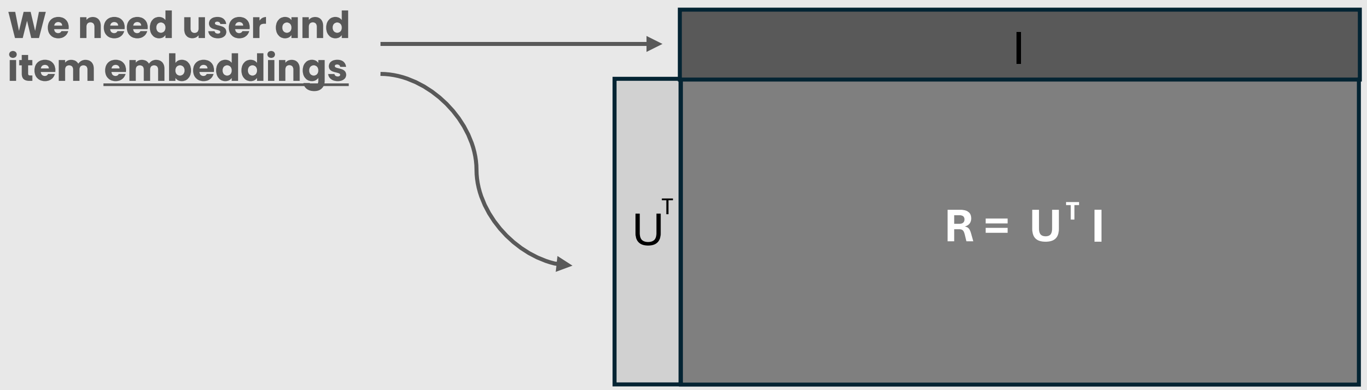
Implementation

PROBLEMS:

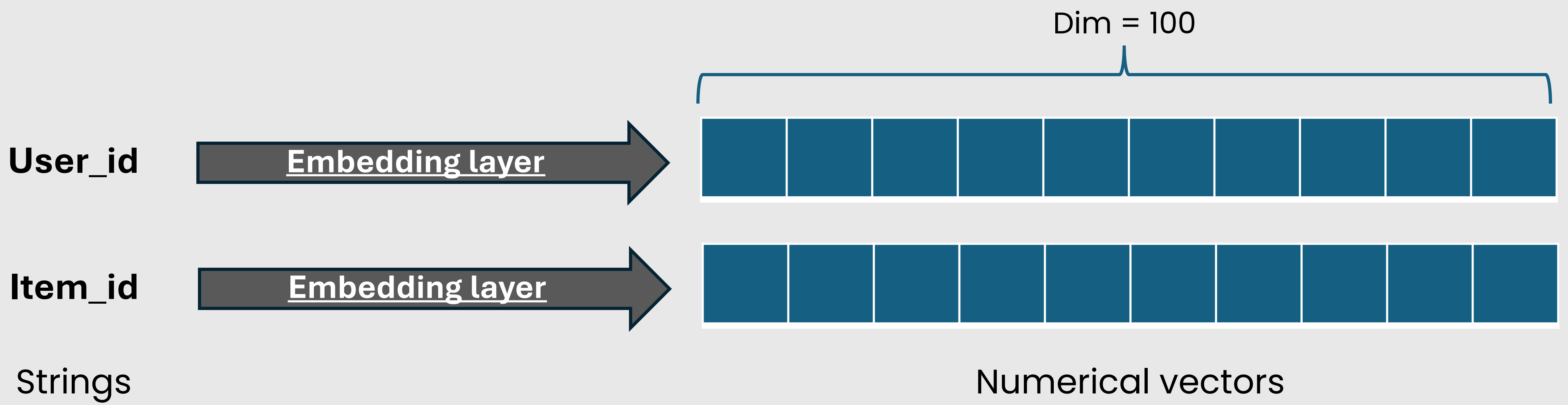
- Matrix R is too large
- How do we predict ratings

SOLUTION: **PMF (Probabilistic Matrix Factorization)**

- *Estimating R by using two low rank matrices* ***U*** *and* ***I***



Learn PMF with Neural Networks



INTERACTION = USER_EMBEDDING • ITEM_EMBEDDING

PREDICTION = INTERACTION + USER_BIAS + ITEM_BIAS

1-dimensional vectors learned by the model with embedding layers

Implementation

Finally we have:

1- User embeddings

2- Item embeddings

3- User x Item rating predictions

WHY DO WE NEED REINFORCEMENT LEARNING?

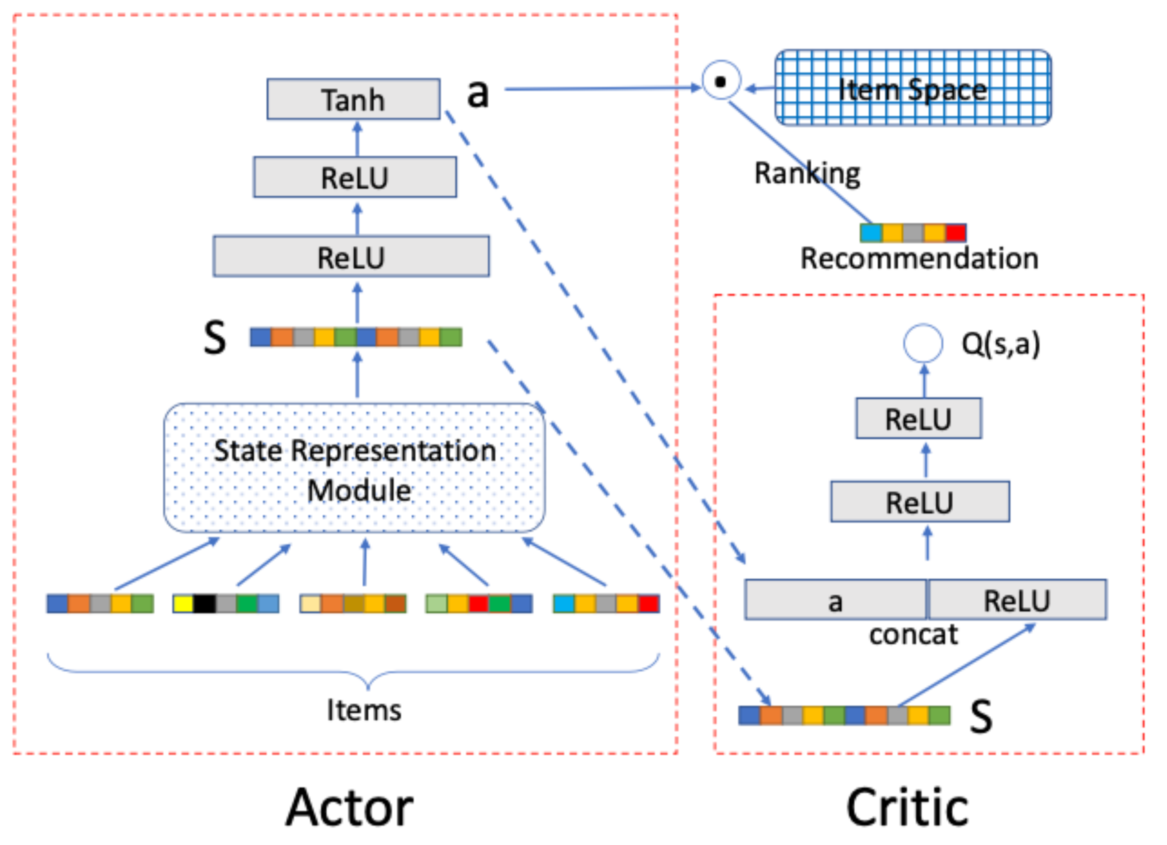
Why do we need reinforcement learning

*PMF recommendations are **STATIC**.*

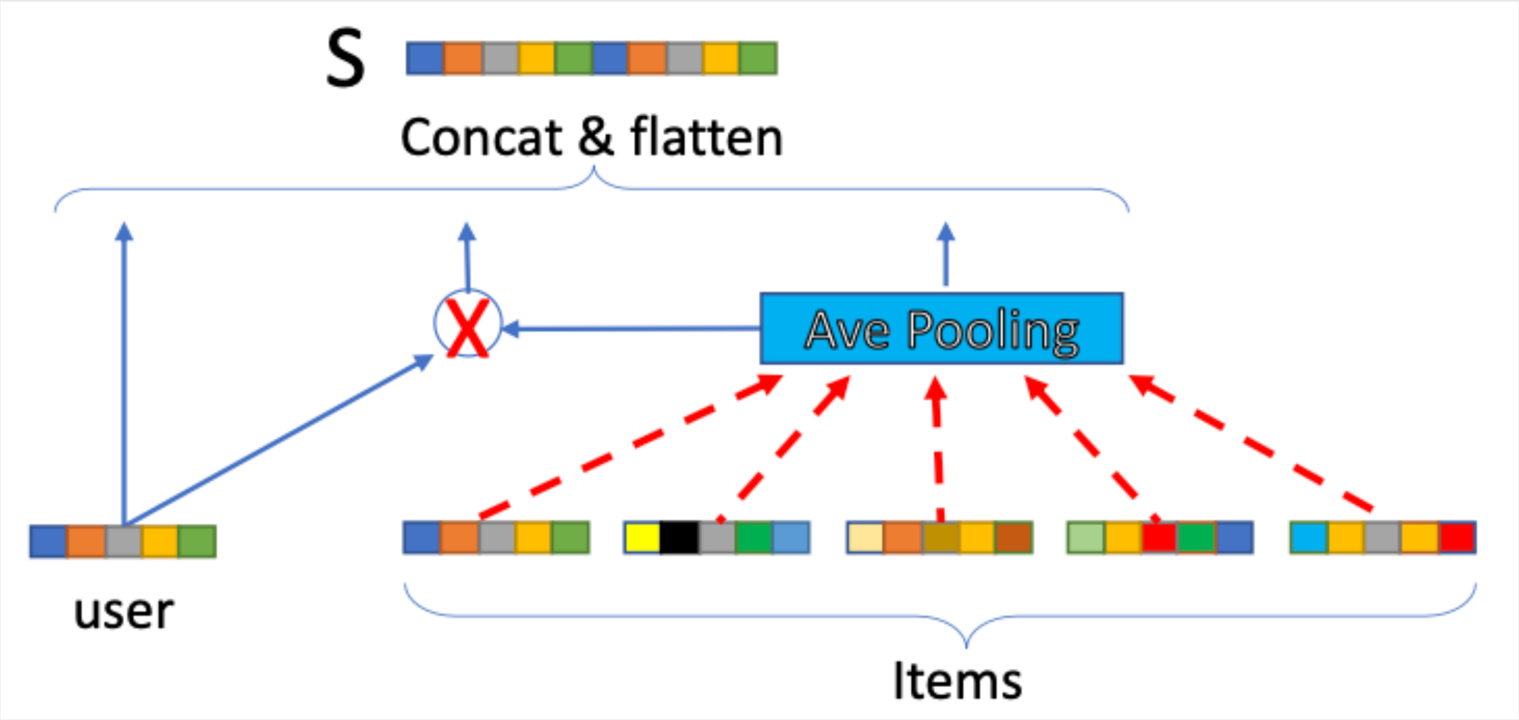
*We want a **DYNAMIC** system that improves recommendations over time according to user preferences.*

Preferences also chage over time

Deep Reinforcement learning based Reccomedation (DRR)



State Representation Module DRR-ave



Two important data structures

- DDR utilizes the user interaction history with the recommender agent as training data.

We need a HISTORY BUFFER H

- In timestep t, the training generate a transition T. Not only we need to store this transition, but we also need to sample a minibatch of N transitions with prioritized experience sampling technique.

We need a REPLAY BUFFER D

REMINDER: A transition is a tuple containing state, action, reward and next state given a particular timestep.

TRAINING ALGORITHM

input : Actor learning rate η_a, Critic learning rate η_c, discount factor γ, batch size N, state window size n and reward function R

1 Randomly initialize the Actor π_θ and the Critic Q_ω with parameters θ and ω

2 Initialize the target network π' and Q' with weights $\theta' \leftarrow \theta$ and $\omega' \leftarrow \omega$

3 Initialize replay buffer D

R is given by the PMF

After initializing Actor and Critic networks, we simply copied their weights to inizialize the target networks

for *session* = *1*, *M* **do**

Each session corresponds to a user

- Observe the initial state s_0 according to the offline log
- **for** t = *1*, T **do**
  - Observe current state $s_t = f(H_t)$, where $H_t = \{i_1, ..., i_n\}$
  - Find action $a_t = \pi_\theta(s_t)$ according to the current policy with ε-greedy exploration
  - Recommended item i_t according to action a_t by Eq. (2)

    $$score_t = i_t a^T$$

  - Calculate reward $r_t = R(s_t, a_t)$ based on the feedback of the user
  - Observe new state $s_{t+1} = f(H_{t+1})$, where $H_{t+1} = \{i_2, ..., i_n, i_t\}$ if r_t is positive, otherwise, $H_{t+1} = H_t$
  - Store transition (s_t, a_t, r_t, s_{t+1}) in D
  - Sample a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) in D with prioritized experience replay sampling technique
  - Set $y_i = r_i + \gamma Q_{\omega'}(s_{i+1}, \pi_{\theta'}(s_{i+1}))$
  - Update the Critic network by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q_\omega(s_i, a_i))^2$
  - Update the Actor network using the sampled policy gradient:

    $$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_t \nabla_a Q_\omega(s,a)|_{s=s_t, a=\pi_\theta(s_t)} \nabla_\theta \pi_\theta(s)|_{s=s_t}$$

  - Update the target networks:

    $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$

    $\omega' \leftarrow \tau\omega + (1-\tau)\omega'$

    Soft update

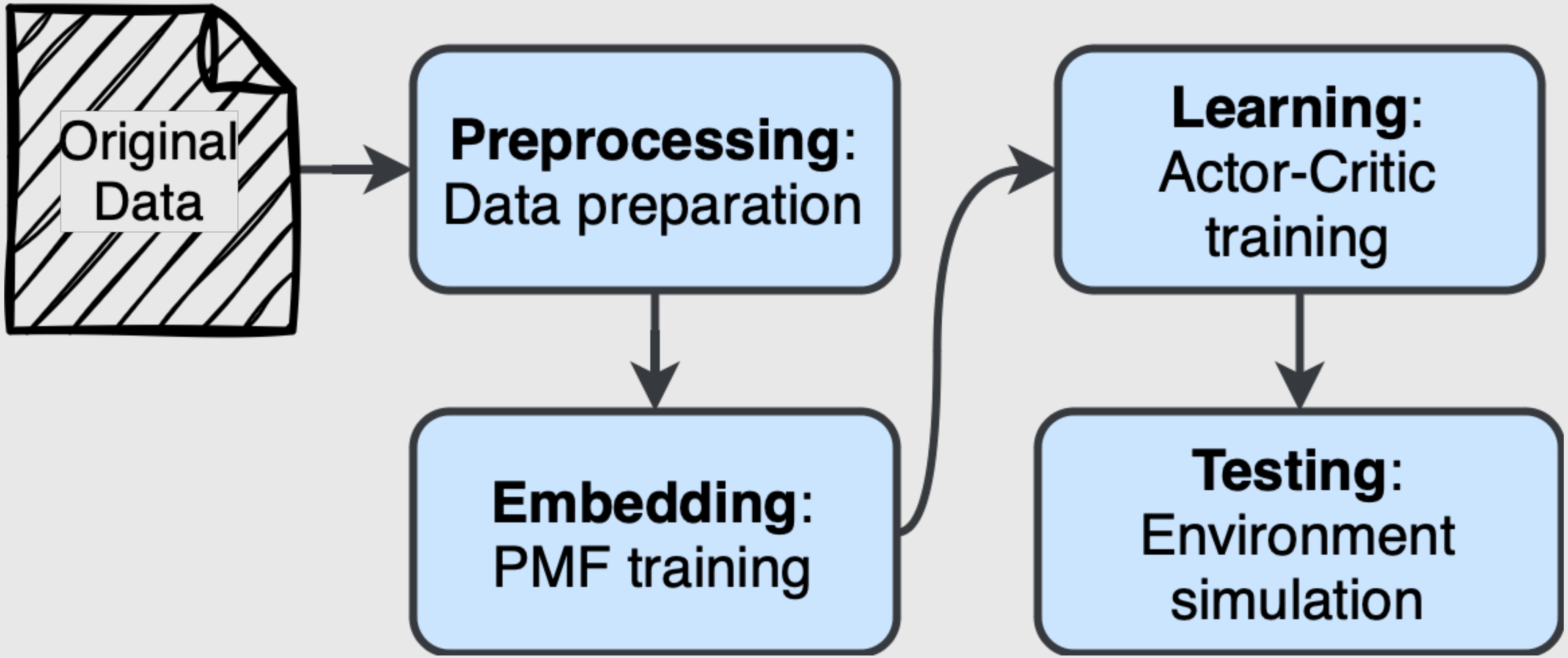
return θ and ω

Experimentation

- **Dataset**: *MovieLens*, ratings by 270000 users for 45000 movies
- **Normalization**: to be used as rewards, ratings $r \in [1,5]$ have been scaled to $R^{\star} \in [-1,1]$, so that positive rewards are given only for ≥ 3 rating values:

$$R^{\star}(s,a) = \frac{1}{2}(r_i - 3)$$

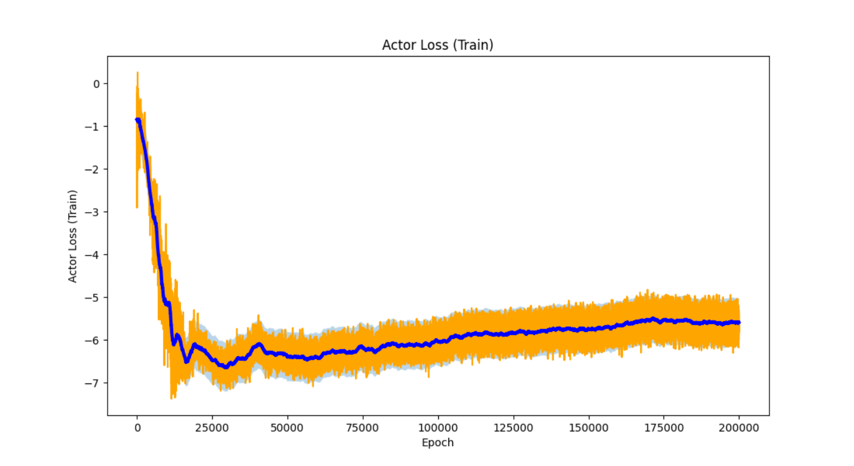
Pipeline

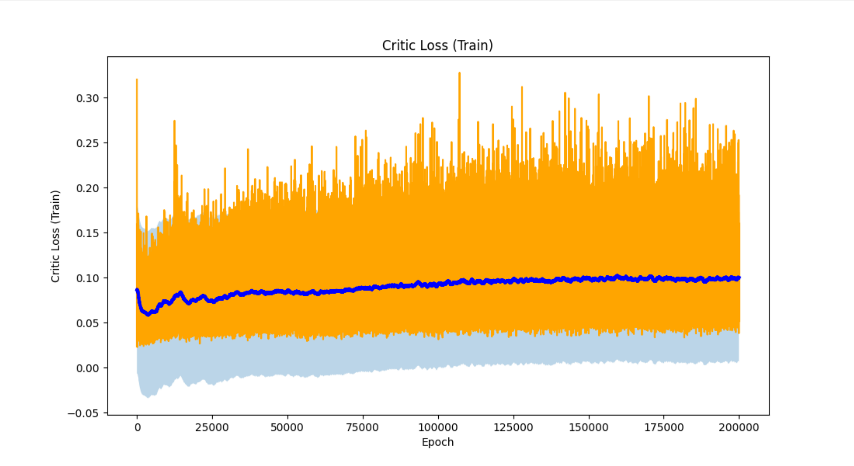

Original Data
Preprocessing: Data preparation
Embedding: PMF training
Learning: Actor-Critic training
Testing: Environment simulation

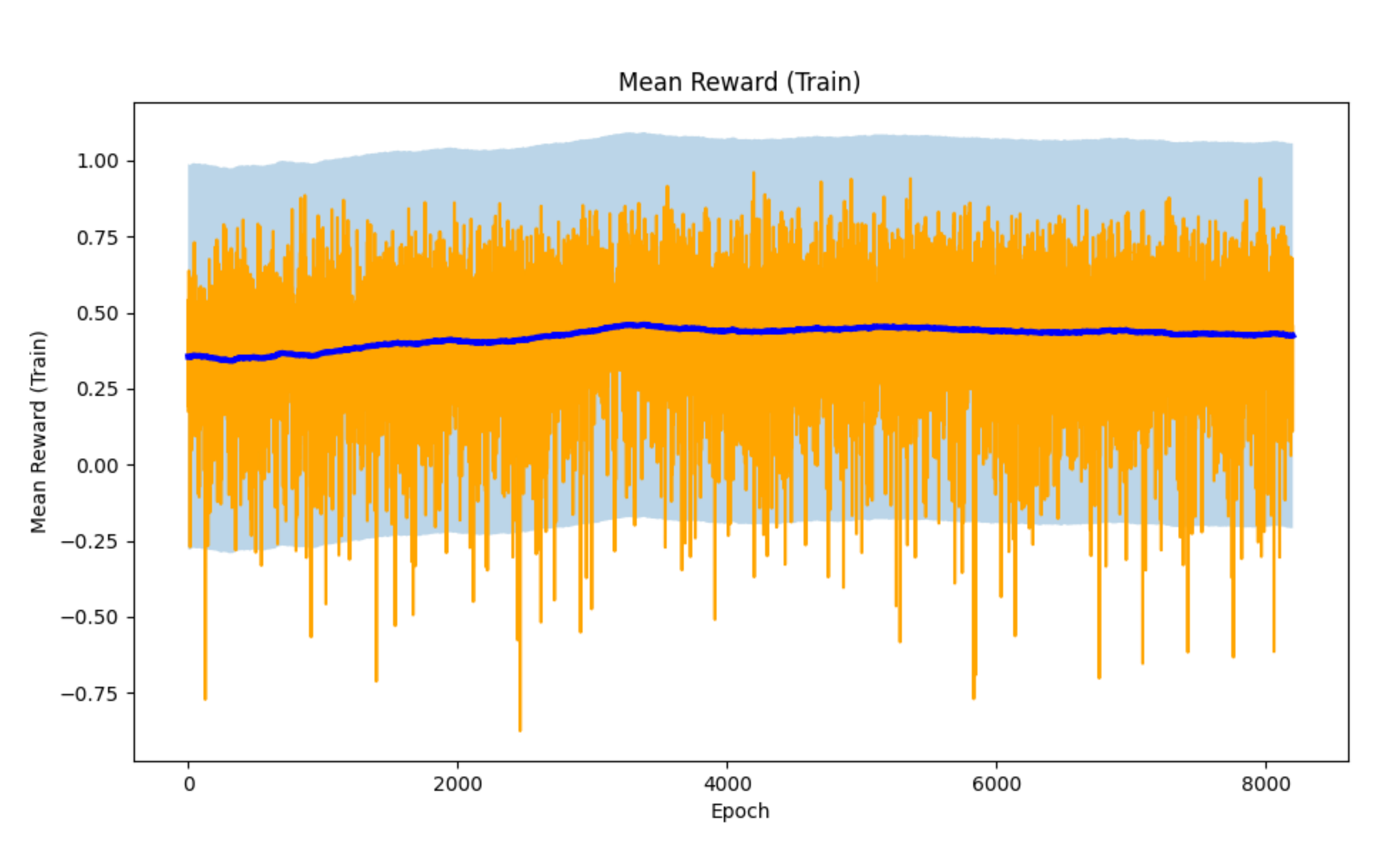
Actor–Critic training

replay buffer $= 200000$, history buffer $= 5$, lr actor $= 0.0005$, lr critic $= 0.001$

$\epsilon_{start} = 1$, $\epsilon_{\text{end}} = 0.1$, $\tau = 0.001$, $\beta = 0.4$, $\alpha = 0.3$, $\gamma = 0.9$







Offline Evaluation

Algorithm 2: Offline Evaluation Algorithm of DRR Framework

input : state window size n and reward function R

Observe the initial state s_0 and item set $\mathcal{I}$ according to the offline log

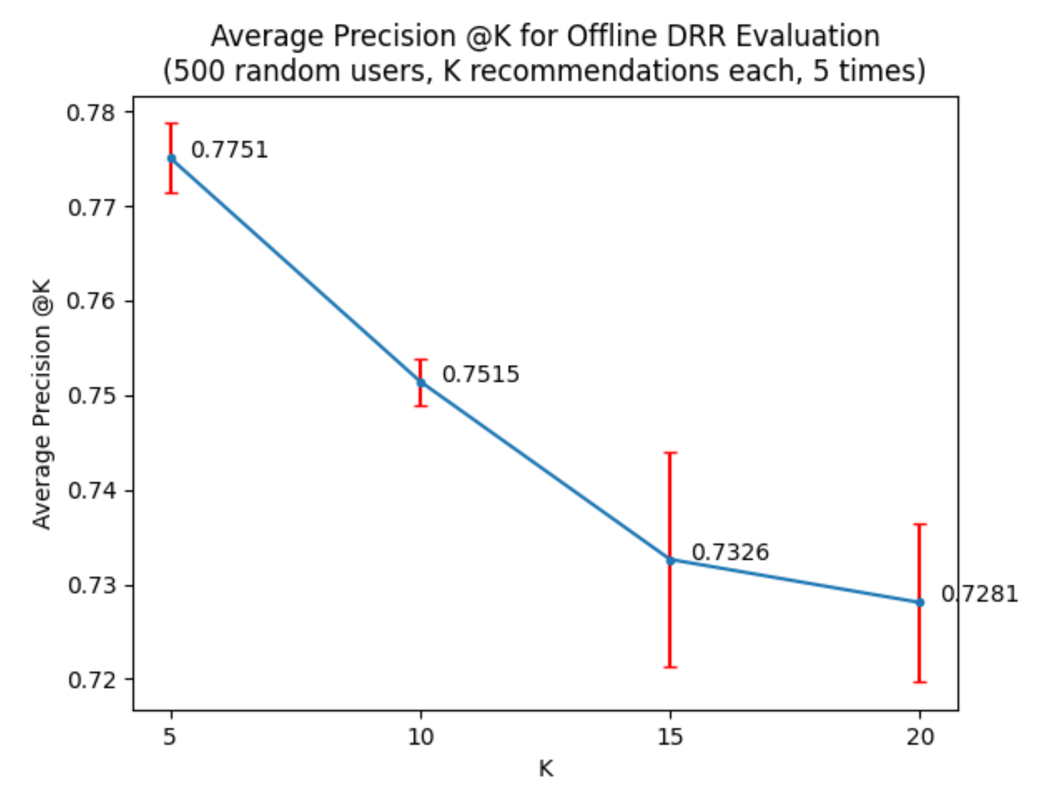
for t = *1*, T **do**

- Observe current state $s_t = \{i_1, ..., i_n\}$
- Execute action $a_t = \pi_\theta(s_t)$ according to the current policy
- Observe the recommended item i_t according to action a_t by Eq. (2)
- Get reward $r_t = R(s_t, a_t)$ from the feedback located in the users' log by Eq. (10)
- Update to a new state $s_{t+1} = f(H_{t+1})$, where $H_{t+1} = \{i_2, ..., i_n, i_t\}$ if r_t is positive, otherwise, $H_{t+1} = H_t$
- remove i_t from $\mathcal{I}$

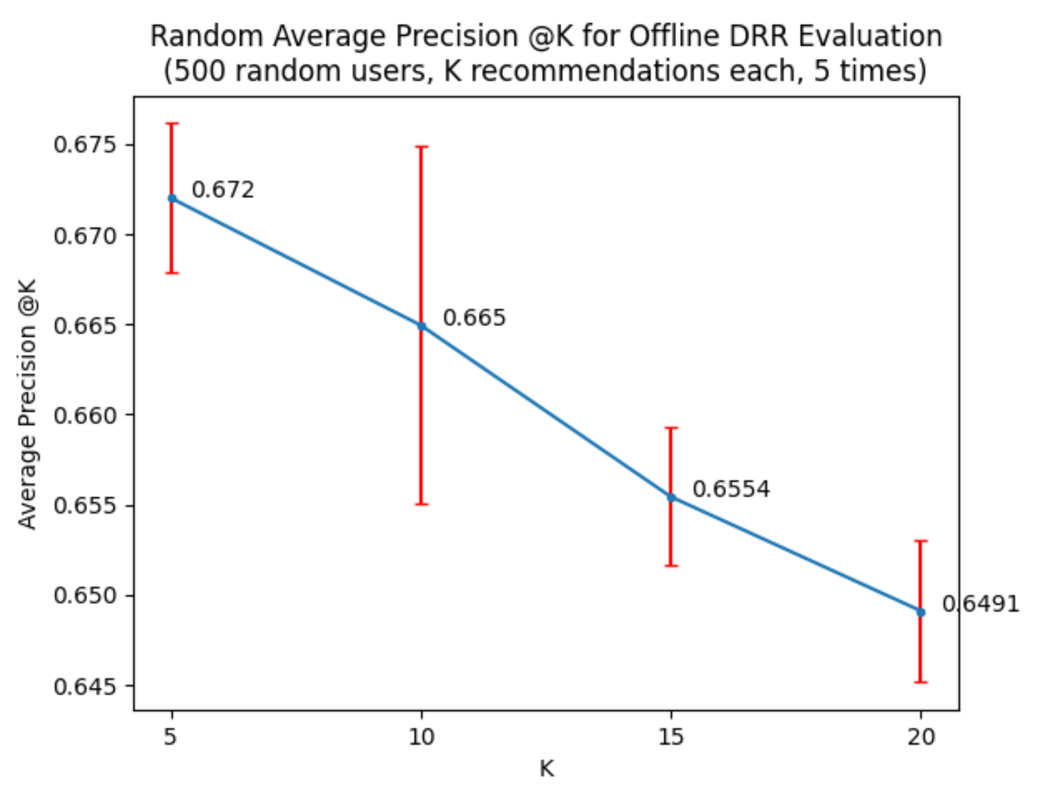
For different episode length $T = [5, 10, 15, 20]$, in the offline evaluation recommendations are made and judged from the set of items the user has rated rather than all candidate items.

The evaluation is done considering **Precision @ K**, the amount of positive predictions made divided by the total amount of predictions made (K) for a user.

Trained model



Randomly initialized model



Trained model precision			
@ 5	@ 10	@ 15	@ 20
0.775 ± 0.004	0.752 ± 0.003	0.733 ± 0.011	0.728 ± 0.008

Randomly initialized model precision			
@ 5	@ 10	@ 15	@ 20
0.672 ± 0.004	0.665 ± 0.010	0.655 ± 0.004	0.649 ± 0.004

Paper precision results	
@ 5	@ 10
0.7693	0.6594

Online Evaluation

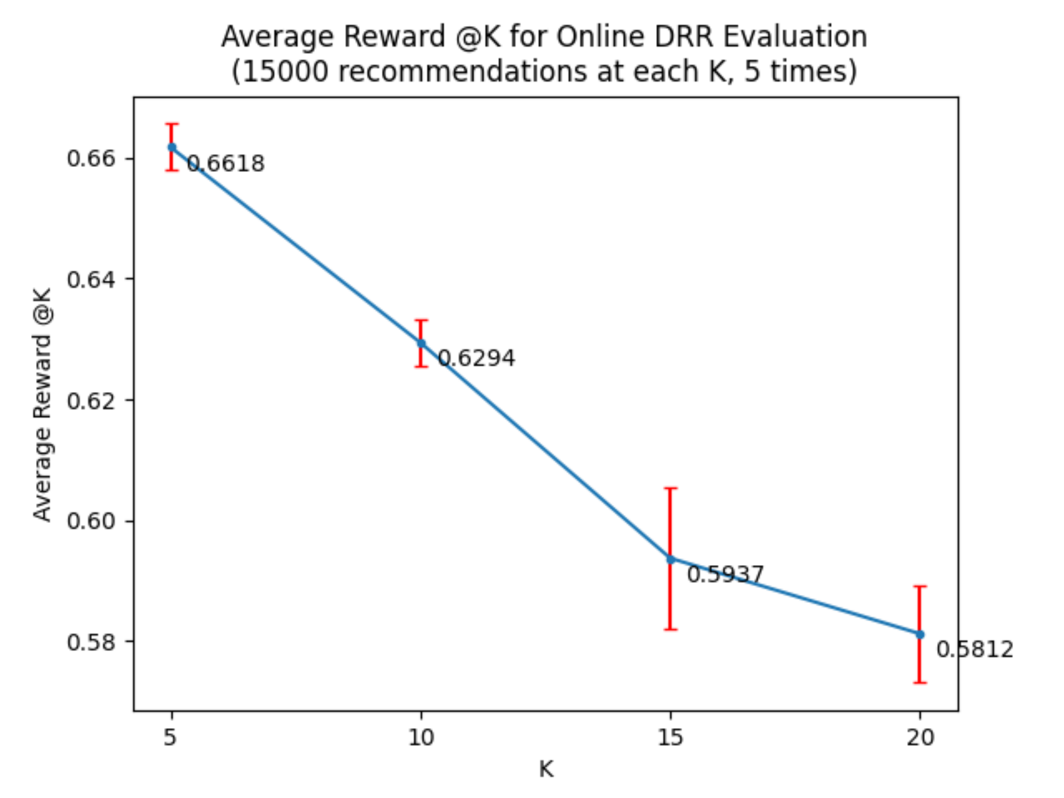
Algorithm 1: Training Algorithm of DRR Framework

input : Actor learning rate η_a, Critic learning rate η_c, discount factor γ, batch size N, state window size n and reward function R

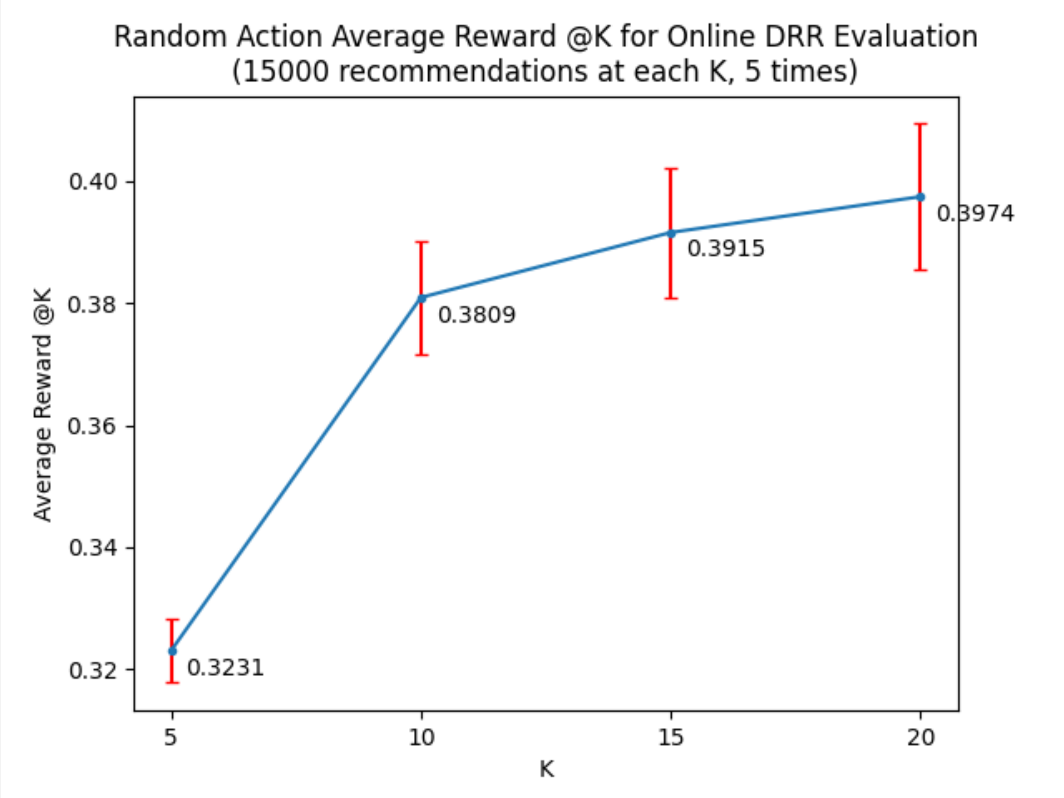
```
Randomly initialize the Actor π_θ and the Critic Q_ω with parameters θ and ω
Initialize the target network π′ and Q′ with weights θ′ ← θ and ω′ ← ω
Initialize replay buffer D
for session = 1, M do
    Observe the initial state s_0 according to the offline log
    for t = 1, T do
        Observe current state s_t = f(H_t), where H_t = {i_1, ..., i_n}
        Find action a_t = π_θ(s_t) according to the current policy with ε-greedy exploration
        Recommended item i_t according to action a_t by Eq. (2)
        Calculate reward r_t = R(s_t, a_t) based on the feedback of the user
        Observe new state s_{t+1} = f(H_{t+1}), where H_{t+1} = {i_2, ..., i_n, i_t} if r_t is positive, otherwise, H_{t+1} = H_t
        Store transition (s_t, a_t, r_t, s_{t+1}) in D
        Sample a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) in D with prioritized experience replay sampling technique
        Set y_i = r_i + γ Q_ω′(s_{i+1}, π_θ′(s_{i+1}))
        Update the Critic network by minimizing the loss: L = (1/N) Σ_i (y_i − Q_ω(s_i, a_i))^2
        Update the Actor network using the sampled policy gradient:
            ∇_θ J(π_θ) ≈ (1/N) Σ_t ∇_a Q_ω(s, a)|_{s=s_t, a=π_θ(s_t)} ∇_θ π_θ(s)|_{s=s_t}
        Update the target networks:
            θ′ ← τθ + (1 − τ)θ′
            ω′ ← τω + (1 − τ)ω′
return θ and ω
```

For different episode length $T = [5, 10, 15, 20]$, the online evaluation follows the training algorithm with the only difference being the networks and their parameters are reset to those of the learned networks before each episode. This is close to a real scenario where users start with the base model and continue training or refining the model with their own interactions. The evaluation is done considering **Avg.Reward @ K**, the average reward per user receiving K recommendations.

Trained model



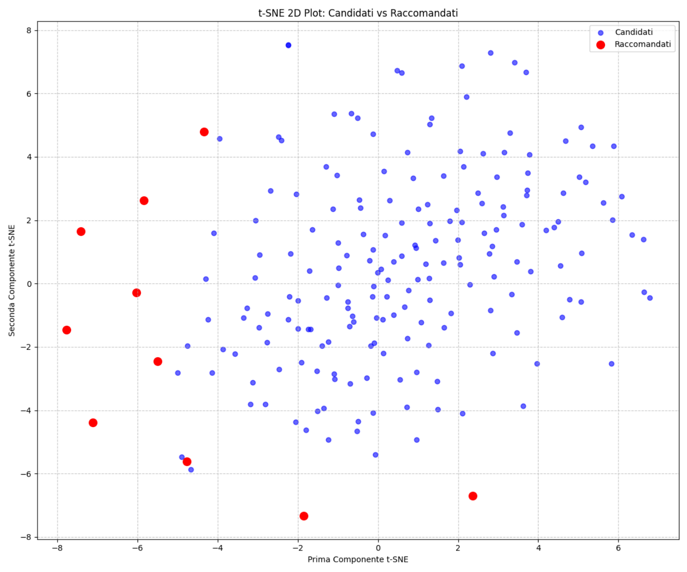
Randomly initialized model

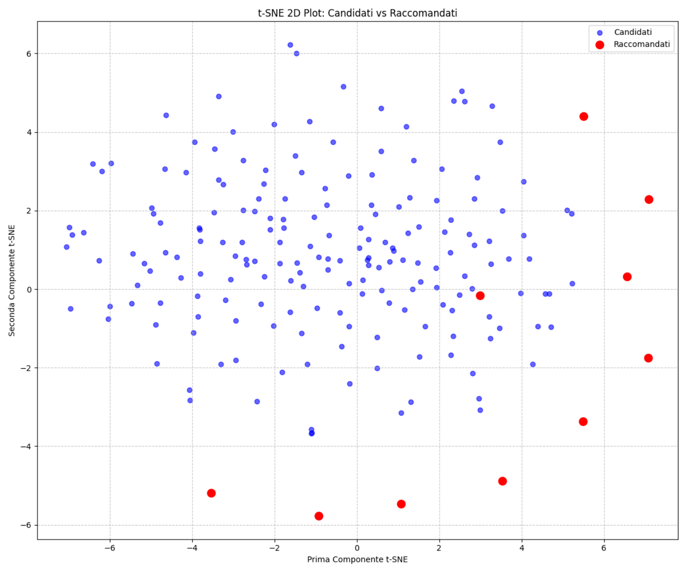


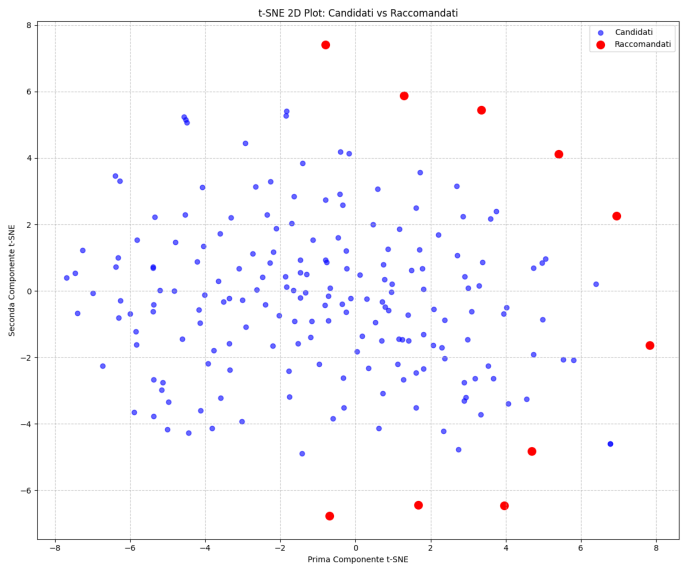
Trained model average reward			
@ 5	@ 10	@ 15	@ 20
0.662 ± 0.004	0.629 ± 0.004	0.594 ± 0.012	0.581 ± 0.008

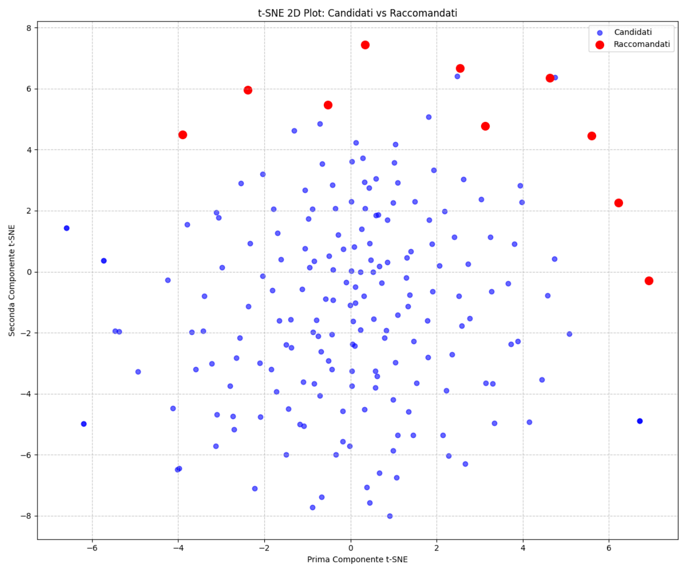
Randomly initialized average reward			
@ 5	@ 10	@ 15	@ 20
0.323 ± 0.005	0.381 ± 0.009	0.392 ± 0.011	0.397 ± 0.009

Recommended items visualization

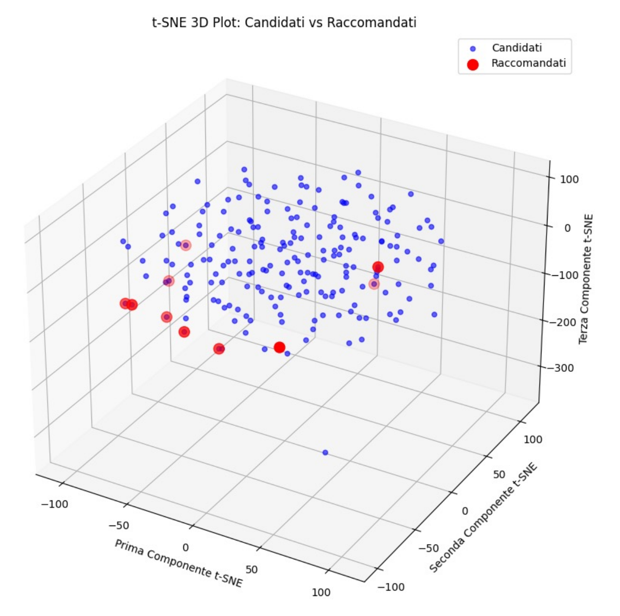


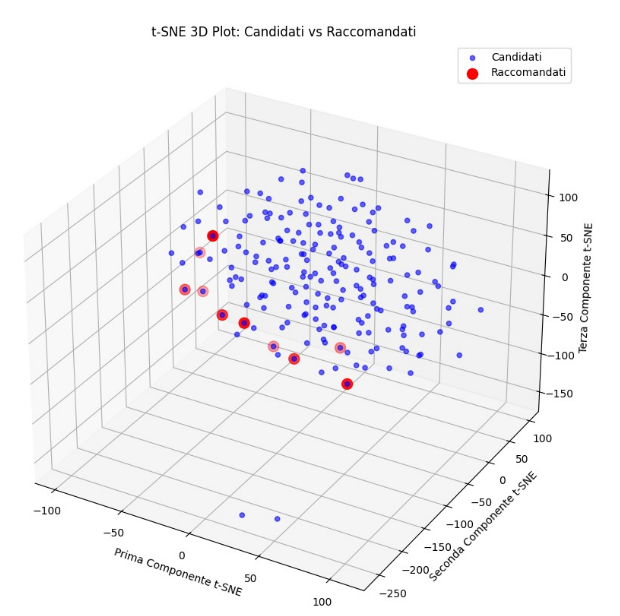


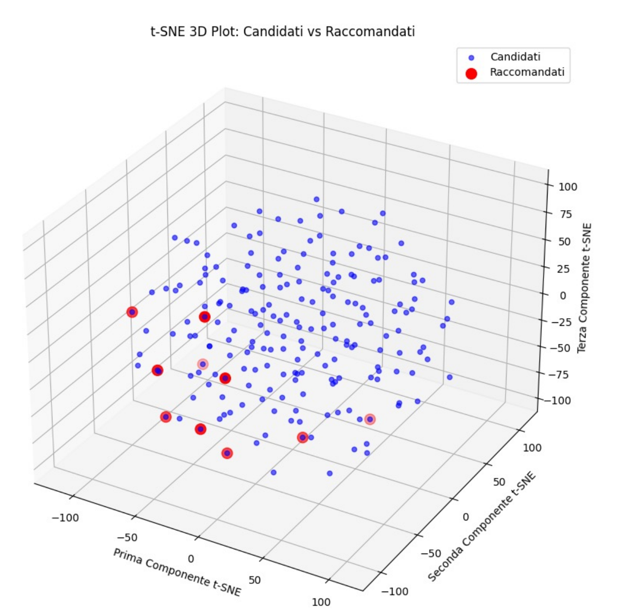


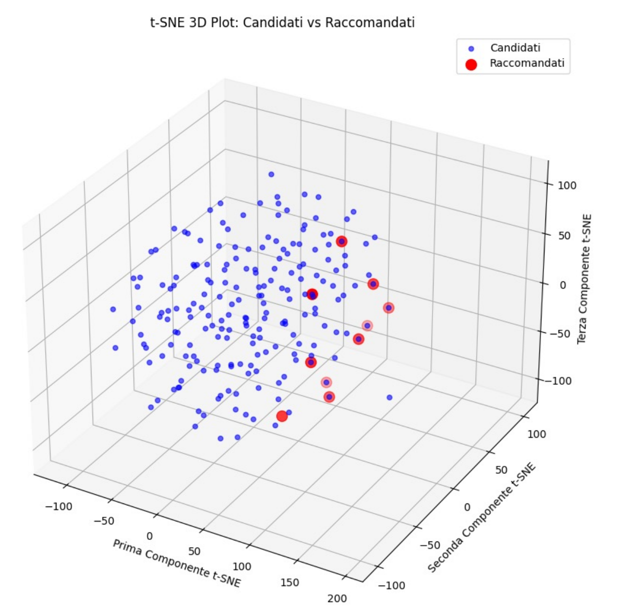


Recommended items visualization









CONCLUSIONS