Deep Reinforcement Learning-based Movie Recommendation System and Comparison Analysis

Amy Tang, Mingyan Xia, Vivian Yan, Christina Yang

DS-GA 3001 Reinforcement Learning Project Presentation

Introduction

Motivations

- Traditional recommendation models, such as collaborative filtering, often fall short when accounting for long-term user engagement and sequential decision-making.
- We plan to leverage reinforcement learning to model recommendation process as a sequential decision problem to provide more personalized and engaging recommendations.

Objectives

- **Develop an RL Environment**: Create a simulation environment based on the **Movielens-100k** dataset that mimics user-movie interaction.
- Movielens: user rating of movies (0-5 star rating)
- **Formulate the Problem**
  - Observations: user vectors
  - Actions: movies available for recommendation
  - States: user history
  - Rewards: movie ratings, watch time
- **Implement RL Algorithms**: Experiment with deep RL algorithms such as Deep Q-Networks (DQN), and Actor-Critic methods.
- **Evaluate Model Performance**: Use comparison analysis with offline metrics to analyze strengths and weaknesses of different RL models.

NYU

Methodology

Action Selection Strategies

- **Epsilon-Greedy Bandit Policy**
  - Select actions randomly with probability and greedily otherwise.
  - A higher epsilon increases exploration, while a lower epsilon favors exploitation.
- **Upper Confidence Bound (UCB) Policy**
  - Encourage exploration by selecting actions with high uncertainty.
  - Prefer arms that could potentially yield high rewards based on both observed performance and uncertainty about the estimate.

Deep RL Model Space

- **Deep Q-Network (DQN)**
  - Incorporate experience replay buffer to stabilize learning.
- **Proximal Policy Optimization (PPO)**
  - Balance between exploration and exploitation with clipped surrogate objectives.
- **Deep Deterministic Policy Gradient (DDPG)**
  - Adopts the actor-critic framework for continuous action spaces.

Model 1: MAB

SimpleMAB

1.Random 2.LinearUCB 3.LinearThompson

4.Greedy 5.GreedyEpsilon

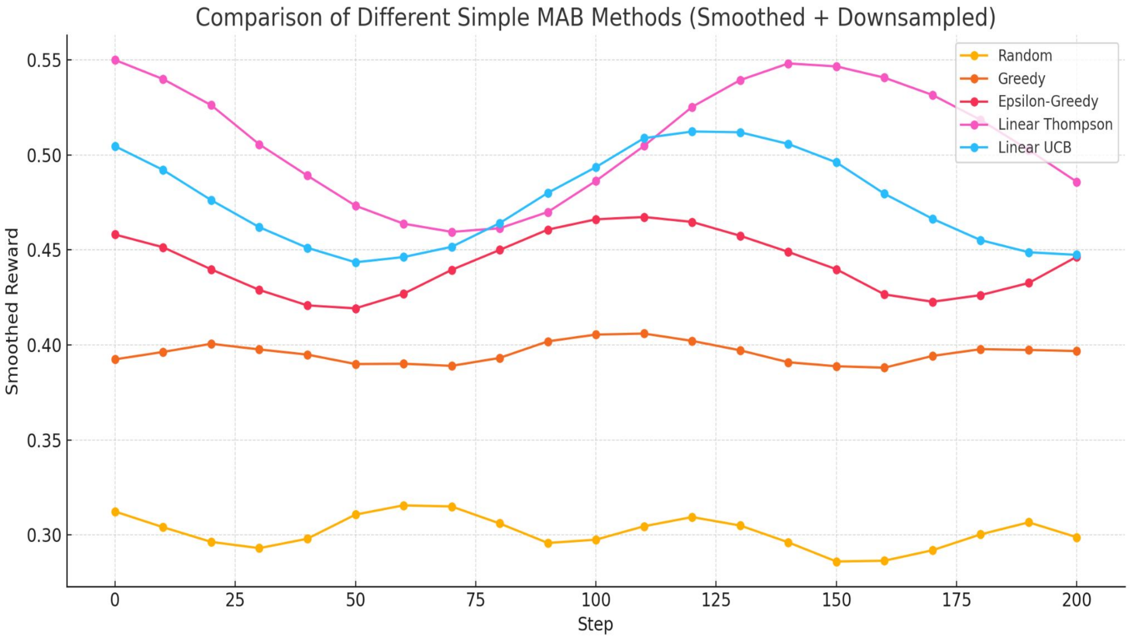
Simple MAB does not utilize user context.

It learns a global best action,

not a personalized ranking.

Without per-user recommendation lists,

Precision@K and Recall@K are not meaningful.



NYU

Model 1: MAB

Contextual MAB

A **Contextual Multi-Armed Bandit** framework is applied for movie recommendation, where user embeddings guide action selection to maximize cumulative reward.
We used a **neural reward model** and **Epsilon-Greedy** exploration strategy, and evaluated via **Top-K metrics**.

Pipeline

Step 1: Load and preprocess MovieLens data
Step 2: Build user embeddings and movie embeddings
Step 3: Create bandit environment
Step 4: Train **bandit agent (reward network + policy)** with exploration (Epsilon-Greedy)
Step 5: Periodically evaluate Top-K metrics on test users

MAB Details

Contextual Embeddings

User Embedding:

Features: Age, Gender, Occupation
Input → Dense layer → 16-dim embedding

During training: Guides the agent's action selection

During evaluation: Recommend personalized movies

Reward Network

The reward network is trained by minimizing the difference between the predicted reward scores and the ground-truth rewards provided by the environment, enabling it to predict user preferences based on user embeddings.

- Input: User embedding (16-dim)
- 2 hidden layers: 256 → 128 units (ReLU + BatchNorm + Dropout)
- Output: Scores for each movie

Purpose: Predict reward (likelihood of user liking the movie)

MAB Result

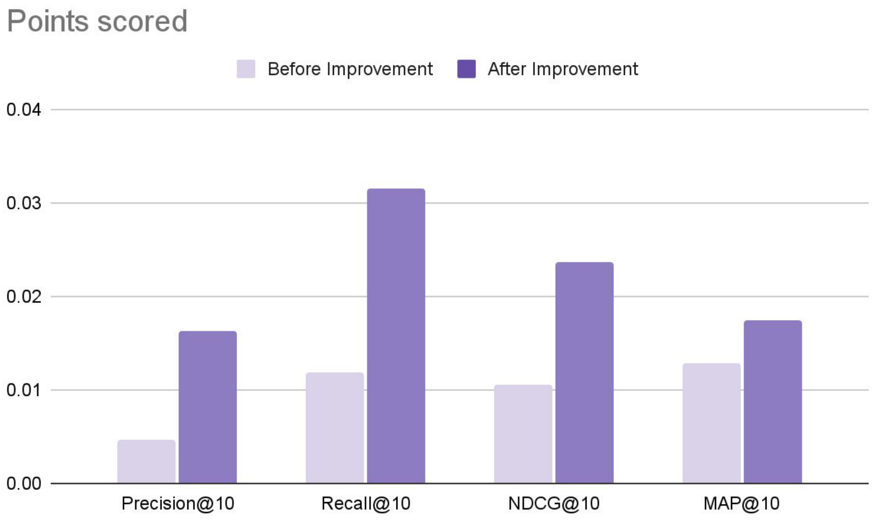
Top-K metrics:
Precision@10: 0.0163 | Recall@10: 0.0315
NDCG@10: 0.0237 | MAP@10: 0.0174

Model Improvement

Reward Shaping: Denser and smoother reward signal
Assign 1.0 for ratings ≥ 4; Assign 0.5 for rating = 3; Assign 0.0 otherwise.

Epsilon Decay: Early exploration, later exploitation
Epsilon starts high (0.3) and decays:
0–10k steps: 0.3
10k–30k steps: 0.1
30k–50k steps: 0.05
50k steps: 0.01

Multi-step Episode: More exploration per user, faster learning
Each user gets 5 consecutive recommendations per episode



NYU

Model 2: DQN

Overview

DQN is a value function approximation method.

- DQN predicts the Q-value for each possible movie action based on the user's watch history to learn at every step from stable buffers of experiences.
- **Experience replay**: reduce correlation between predicted v.s target values to make the training data more i.i.d.

Pipeline

- Create a **custom RL environment** using OpenAI Gym: recommend movies to users based on their past viewing history and ratings.
- Train a **DQN agent** using Keras-RL: with improvements on model structure, optimizer and exploration policy.
- **Evaluate** model performance: using precision, recall, NDCG and MAP.

DQN Details

Environment

- **States**: a history of watched movies, where 10 past movies are remembered.
- **Actions**: discrete - recommend a new available movie by selecting its index.
- **Rewards**: +1 if the user liked the recommended movie with rating>=4, otherwise 0.

DQN Model

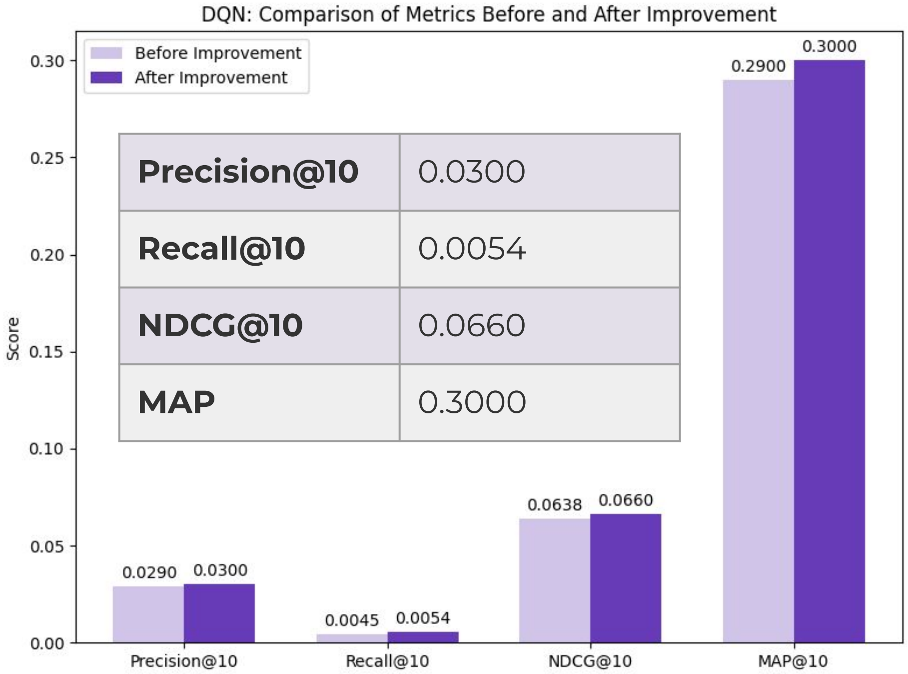
- **DQN agent**: use movie embeddings as input, add two dense layers, use linear activation for final layer.
- **Configuration**: apply a standard replay buffer with 50000 experiences, epsilon greedy exploration with linear decay.
- **Compilation**: use double DQN to avoid overestimating Q-values with Adam and mean absolute error.

NYU

DQN Result Comparison

Model Improvement

- **Model structure**: Upgrade the network to be deeper (+1 layer) and wider (double neurons).
- **Exploration policy**: slowly decay epsilon to allow more exploration early.
- **Optimizer**: On top of a simple Adam, use lower learning rate (from 1e-3 to 5e-4) and add clipnorm (=1.0) to stabilize updates.



Precision@10	0.0300
Recall@10	0.0054
NDCG@10	0.0660
MAP	0.3000

NYU

Model 3: DDPG

DDPG Overview

Deep Deterministic Policy Gradient (DDPG) is an **off-policy actor–critic** method that learns a deterministic policy and a Q-value function simultaneously, enabling effective control in continuous action spaces.

Applying DDPG to MovieLens

Applying DDPG to MovieLens frames personalized movie recommendation as a **continuous-action Markov Decision Process**, where a GRU-based **Actor** proposes next-movie embeddings and a **Critic** evaluates them via Q-values, trained off-policy with experience replay and soft target updates.

NYU

DDPG Details

Data Pipeline

- Load & merge ratings + movie titles
- Create per-user chronological histories
- Split users into train/test (no overlap)
- Save state / action_reward CSV for replay buffer

Movie Embeddings

- Train a neural autoencoder-style model to learn dense vectors for each movie
- One-hot user history → hidden layer → softmax reconstruction
- Extract and save the first-layer weights as item embeddings

Environment Simulator

Define MDP where:

- State = a list of movies that user has rated
- Action = a list of movies recommended to the user
- Reward = whether the user liked the recommendation or not

NYU

DDPG Details (cont.)

Actor Network

- GRU processes variable-length history → final hidden state
- Dense projection → sequence of ra_length action-embedding vectors
- Online & target copies with soft-update τ

Critic Network

- GRU on state + concatenated action embedding → two FC layers → Q-value
- Online & target critics, MSE loss on TD target

Experience Replay

- Store (state, action, reward, next_state) transitions
- Sample mini batches for off-policy updates

NYU

DDPG Details (cont.)

Training Loop

- Alternate updates:
  - Critic: minimize MSE against target critic
  - Actor: ascend ∇ □$Q \circ \mu$ via sampled action gradients
- Soft-update target networks each step

Evaluation & Metrics

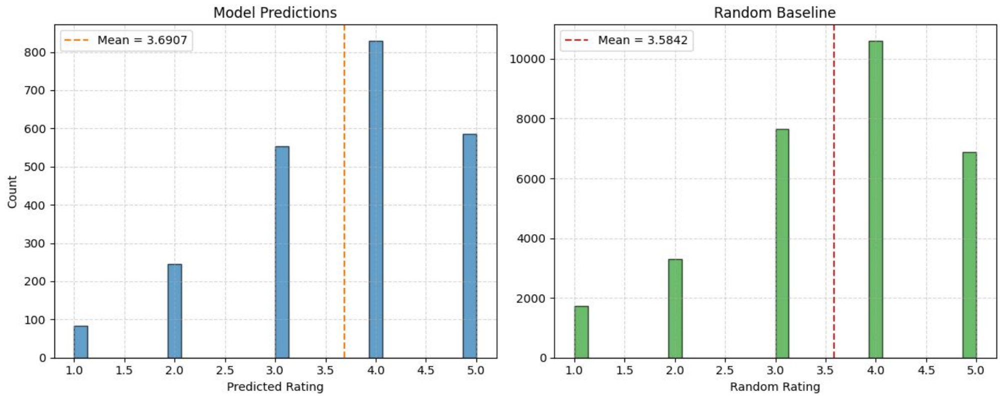
- Predict recommendations for held-out users
- Compute ranking metrics on train/test sets

DDPG Result

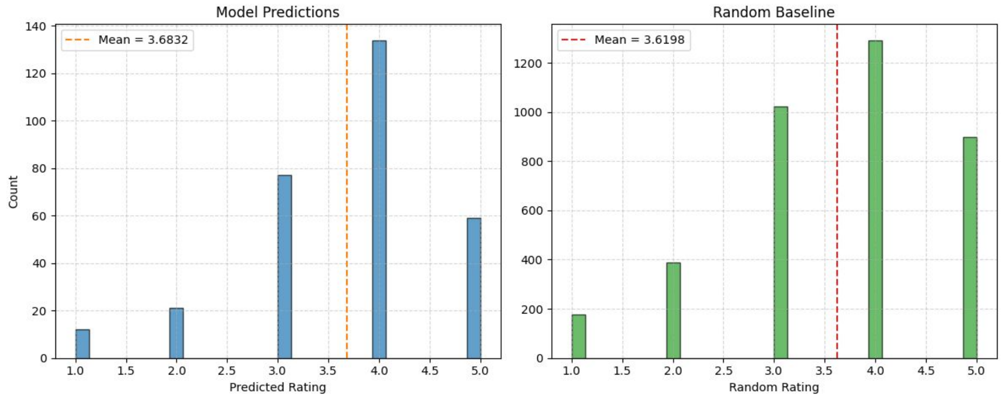
Precision@10	0.0751
Recall@10	0.0202
NDCG@10	0.0879
MAP	0.0114

- Better performance than the random baseline.
- Overall ranking quality is still modest

Training Set



Test Set



Model 4: PPO

Overview

Proximal Policy Optimization (PPO) is an on-policy actor-critic algorithm that combines the benefits of policy gradients and value function baselines. It optimizes a surrogate objective with a clipped probability ratio to prevent overly large policy updates, ensuring stable learning.

Applying PPO to MovieLens data

We treat movie recommendation as a sequential decision problem where each step is “recommend one movie and observe feedback.”

- **State**: User and movie features — normalized user mean rating, normalized movie mean rating, one-hot genre vector, age-bucket one-hot, occupation one-hot, gender one-hot
- **Action**: A discrete choice among {1,2,3,4,5}, interpreted as the predicted user rating for the current movie
- **Reward**: A smooth, bounded signal with perfect predictions scoring 1.0 and larger errors penalized smoothly

NYU

PPO Details

Pipeline

- Define MovieRecEnv that maps each (user, movie) pair to a feature vector and accepts a discrete action (predicted 1 - 5) with a smooth reward
- Launch four parallel envs under DummyVecEnv, wrap in VecNormalize to standardize observations
- Train PPO for 200k timesteps
- Evaluate by measuring the four metrics

PPO Hyperparameters

Learning rate	1×10^{-4}
Discount	0.98
Rollout length (n_steps)	256
Batch size	128
Clip range	0.1
Entropy bonus	0.01

NYU

PPO Result

Reward-based Performance

Average Episode Return:

- Trained for 200k timesteps on 4 parallel envs
- Evaluated over 10 full episodes

Average reward over 10 episodes: 315.54

Ranking Metrics @ 10

Precision@10	0.1839
Recall@10	0.1332
NDCG@10	0.1893
MAP@10	0.0465

NYU

Comparison Analysis

	Precision@10	Recall@10	NDCG@10	MAP@10
Contextual MAB	0.0163	0.0315	0.0237	0.0174
DQN	0.0300	0.0054	0.0660	0.3000
DDPG	0.0751	0.0202	0.0879	0.0114
PPO	**0.1839**	**0.1332**	**0.1893**	**0.0465**

NYU

Results Explanation

- Why might PPO perform better than others?
  - **Sequential Credit Assignment**:
    - Unlike a contextual bandit, PPO treats recommendation as a multi-step process
    - Its actor-critic architecture explicitly learns to maximize long-term cumulative reward rather than immediate gains
    - It back propagates reward signals through entire trajectories of recommendations, capturing patterns in user behavior over time
  - **Stable, Constrained Updates**:
    - The clipped surrogate objective in PPO prevents any single update from moving the policy too far —> more reliable learning on noisey and sparse feedback
    - DQN's value-based updates or DDPG's continuous control updates on a fundamentally discrete task
  - **Natural Fit for Discrete Ratings**:
    - PPO's discrete action head directly models the 5-point rating scale
  - **Variance Reduction via Critic**:
    - PPO's learned value function provides low-variance advantage estimates, which is more sample-efficient than pure policy-gradient or off-policy Q-learning approaches

Conclusion & Future Steps

- PPO's on-policy actor-critic with clipped updates consistently outperforms bandits, DQN, and DDPG on ranking metrics
- The discrete action head naturally models 5-star ratings, while the value baseline reduces variance and speeds convergence
- Parallel VecNormalize environments deliver stable, well-scaled observations
- Future steps:
  - Experiment with click-through, watch-time, or diversity-aware reward functions with reward engineering
  - Scale up to 1M MovieLens dataset
  - Integrate collaborative-filtering or graph-based embeddings and tune network architectures

References

- Reinforcement Learning-based Recommender Systems using TF-Agent and MovieLens Dataset: https://medium.com/@yuchengtsai84/reinforcement-learning-based-recommender-systems-using-tf-agent-and-movielens-dataset-ebbf40b3a1a2
- Exploration in Interactive Personalized Music Recommendation: A Reinforcement Learning Approach: https://arxiv.org/abs/1311.6355
- Playing Atari with Deep Reinforcement Learning: https://arxiv.org/abs/1312.5602
- Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor: https://arxiv.org/abs/1801.01290
- Deep Reinforcement Learning for List-wise Recommendations: https://arxiv.org/abs/1801.00209
- Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling: https://arxiv.org/abs/1810.12027
- Reinforcement learning based recommender systems: A survey: https://arxiv.org/abs/2101.06286