Chapter 2

Background

2.1 Literature Review

2.1.1 Literature Review for Recommender systems

Recommender systems are widely used in various fields currently, including e-commerce items [61, 101, 109, 97, 54], healthcare [8, 7], news [134, 51], travel [90], Internet of Things (IoT) [56], etc.

Collaborative filtering systems [83] constitute the popular recommender system method with the the core assumption of the that the past preference behaviors of users guide their future behaviors. Collaborative methods are divided into two types: memory-based or model-based. Memory-based CF [83] utilizes the user's past ratings to find the similarity between the items or users, i.e., item-based or user-based types. Model-based CF [9, 49] make use of machine learning methods such as clustering , SVD, PCA, Bayesian analysis, Matrix factorization methods, etc., for predicting user's ratings. Probabilistic matrix factorization [29] techniques have been effective on large and sparse data sets. further, the authors presented the Bayesian probabilistic matrix factorization model [81], adding Bayesian analysis to PMF, increasing the accuracy. Further, the authors presented the Bayesian probabilistic matrix factorization model [81], adding Bayesian analysis to PMF, in-

creasing the accuracy further. Knowledge-based systems [1] propose items based on specialized domain knowledge about how certain item attributes suit the user's needs and preferences, as well as how the item is beneficial to the user. Demographic recommender systems [104] categorize users or objects based on personal traits and further provide recommendations based on demographic categorizations. Hybrid recommender systems [14] generally combine two or more recommender system techniques to generate better recommendations, such as content-based and collaborative filtering methods or multiple collaborative filtering systems [49]. In [104, 105], authors proposed a hybrid recommender system utilizing demographic and feature information of users and items. The demographic recommender approach, [104], utilized demographic information about users and items was proposed generating more accurate predictions for user-based and item-based CF.

Recommender systems utilizing deep learning frameworks have been applied recently utilizing neural network techniques such as Autoencoder, RBM, CNN-RNN deep belief networks, etc., to provide recommendations. Autoencoders have been utilized for collaborative filtering systems recently [3]. In [114], authors proposed the Collaborative Denoising AutoEncoder (CDAE), making use of denoising autoEncoder [103], to provide recommendations by learning hidden structures that can approximate user's ratings given past ratings as inputs. In [61], authors proposed a deep hybrid recommender system that utilized neural networks to extract user and item features from side information. Deep belief networks based music recommender method was proposed in [101]. A restricted Boltzmann machine-based recommender system for large datasets for proposed in[82] utilizing multi-layered graph model. A Collaborative Deep Learning (CDL) [106] method with a hierarchical Bayesian model for recommendation was proposed for addressing sparsity issues utilizing the auxiliary item information.

Reinforcement learning(RL) based recommender systems have been proposed recently to recommend web pages, travel itineraries, books, news, e-commerce items, etc. Policy-based methods [132, 31, 42] are effective in large state-action space as

they directly evaluate the policy without the need to calculate the value over all the actions and thus are preferred over value-based methods. Deep RL-based methods integrating with deep learning framework [134, 132] have also been applied recently in Recommender Systems. A Deep Deterministic Policy Gradient (DDPG) based algorithm was proposed in [58] addressing large action space issue that arises in RL-based settings for recommender systems.

Another approach Policy Gradient for Contextual Recommendation (PGCR) [72] was proposed for recommendations exploiting the contextual information and utilized REINFORCE algorithm to train the agent. A policy gradient approach based on REINFORCE algorithm was proposed in [108] for explainaible recommmendation. In [20], authors proposed natural policy gradient based approach for top-k recommender task utilizing off policy learning, and performed experiments over youtube datasets.

WebWatcher [95], uses Q-Learning to guide visitors to their chosen pages on the World Wide Web approximating pages as states, hyperlinks as actions, and calculates rewards based on the similarity of the pages and user profiles. In [91], authors proposed a travel based recommender system based on the Q learning method exploiting a linear function for ranking trips based on different attributes such as trip price, location duration, etc. A recommender system [85] based on MDP approach was proposed modeling recommendation as a sequential optimization task.

A page-wise recommender system was proposed in [132] that utilized deep reinforcement learning and real-time user feedback to present a group of recommended items on a page with appropriate display In [133] authors proposed an interactive recommender system that utilised deep reinforcement learning framework utilising both positive and negative feedback during learning. Further, Multi-armed bandits based approaches have also been proposed for recommender systems. LinUCB [52] adapted news article recommendations with a contextual bandit approach where the algorithm utilized contextual information related to the users-articles for rec-

ommending articles to users representing news context as feature vectors, and the click-through rates as the payoffs.

2.1.2 Literature Review for Ranking in search systems

Ranking is the key problem in information retrieval systems [4]. It has numerous applications in different domains such as information retrieval, recommender systems, learning to rank, email routing, sentiment analysis, ad placement, etc. [39]. The most common application of ranking is document retrieval which refers to the process of retrieving relevant documents from a collection based on a user's query or information need such as in web search engine. Learning to Rank (LTR) is a machine learning approach used for document retrieval to automatically construct a ranking model for a set of items [60]. Learning to Rank based approaches are generally classified into pointwise [26], pairwise [12, 45] or list wise methods [13, 17]. Pontwise solutions for the ranking problem are often classification or regression based techniques that are limited to predicting a class or relevance label for the input set of documents. For pointwise techniques, any general classification or regression methodology could be used. To perform classification, pairwise algorithms take two documents of varied significance and concurrently learns a loss function. Listwise approaches, on the other hand, use the full list of elements as instances throughout the learning process and then do prediction over the list using the learned model. Further, different machine learning based solutions have been proposed that optimize the evaluation metric such as Precision, NDCG, etc. directly and thus learn the ranking function during training [18, 98, 124, 100, 120]. Support vector machine based approaches such as SVMMAP [124], SVMNDCG [18] have been proposed for the ranking task that optimize various information retrieval metrics, MAP and NDCG, respectively. In [117] , the authors proposed Boosting algorithm ADARANK, for Learning to Rank task by optimizing the NDCG and Precision metrics. In [50], bandit-based approach was proposed based on the cascade model for web page ranking task. Further, Deep Learning based approaches

for learning to rank have been proposed utilizing neural network structure for the task[38, 79, 73, 16].

Reinforcement learning based approaches [94] have been recently applied to the ranking task [110, 85, 34, 115]. In [110], the authors proposed a reinforcement learning approach for the learning to rank problem using a policy gradient algorithm, improving the performance over the NDCG metric. In [98], authors proposed a Multi-Armed bandit solution that utilized user click behavior for ranking web pages. A dual agent bandit game [63] modeled as Parially observed MDP was proposed for the session search task.

In [34], the authors proposed monte carlo based exploratory algorithm using tree search. In [122], authors introduced a duelling bandit-based online learning framework for real-time learning via implicit feedback and utilising pairwise comparisons. Further, other reinforcement learning based approaches have been extensively applied in ranking tasks and search result diversification [115, 59]. In [125],authors proposed a multi-page search scenario task formalizing the Learning to Rank problem as a Markov Decision Process. In [127], a log document based model was proposed for content-free document ranking was formulated as a POMDP utilizing the user's click-behaviors. In [107], work aimed at reducing the variance in Online Learning to Rank was proposed. A multi-agent RL based approach for ranking task was proposed in [139], modeling documents as agents and ranking process as an interactive MDP session between the agents. In [139], a Reinforcement Learning based framework based on natural policy gradient for the personalized search was proposed. In [121], authors proposed RML, a reinforcement learning framework for relevance feedback optimizing different metrics based on REINFORCE algorithm. In [140], a multi-agent Reinforcement Learning approach based on the REINFORCE algorithm was proposed, in which documents were modelled as RL agents and the ranking process was formulated as an interactive MDP amongst the agents. In [32], authors presented a multi-scenario ranking approach as a POMDP-based framework modeling the problem as a multi-agent decision problem.

2.2 Deep Reinforcement Learning

Deep learning is a subfield of machine learning that focuses on training artificial neural networks to learn and perform tasks without explicit programming. It is inspired by the structure and function of the human brain, where neurons work together to process information. The term "deep" in deep learning refers to the depth of the neural networks used in these models. Deep learning models are composed of multiple layers of interconnected artificial neurons, also known as nodes or units. These layers allow the network to learn progressively complex representations of the input data as it flows through the network. Deep learning has achieved remarkable success in various domains, including computer vision, natural language processing, speech recognition, and reinforcement learning. Some of the most popular deep learning architectures include Convolutional Neural Networks (CNNs) for image processing, Recurrent Neural Networks (RNNs) for sequential data, etc.

Deep Q Networks (DQNs) are a class of deep learning models used in reinforcement learning, specifically for solving problems in the context of Markov Decision Processes (MDPs). DQNs are an extension of the Q-learning algorithm, which is a popular and powerful method for training agents to make decisions in an environment to maximize cumulative rewards.

Q-learning utilizes Q-table to store the Q-values, which represent the expected future rewards for each state-action pair in the environment. However, this approach becomes infeasible for large state spaces, as the Q-table grows exponentially with the number of states and actions.

Deep Q Networks solve this problem by using a deep neural network as an approximation of the Q-value function. Instead of storing values in a table, the DQN takes the state as input and outputs Q-values for each action. This neural network architecture allows the agent to learn a continuous function that can generalize across similar states and actions, making it more scalable and efficient for complex environments. Complex environments in reinforcement learning generally refer to scenarios

that present significant challenges and intricacies for an RL agent to learn effective policies such as high-dimensional state and action spaces, partial observability, non-stationarity etc [30, 2].

Deep Reinforcement Learning (DRL) incorporates the functionality of Deep learning in reinforcement learning algorithms. They can process a very large input space and have been used to solve a diverse range of complex decision problems that were previously too complex for traditional RL techniques. Deep RL techniques have been applied to various new applications in domains such as robotics, finance, video games, natural language processing, computer vision, etc. Deep reinforcement learning algorithms utilize a deep learning framework for solving complex problems which represent the policy or value functions as a neural network and integrate them with RL techniques, thus creating specialized algorithms that perform well in this environment. Deep reinforcement learning agents are composed up of a deep neural network policy that maps an input state to an output action and an algorithm that keeps this policy up to date. Popular algorithms include deep Q-network (DQN), deep deterministic policy gradient (DDPG), soft actor critic (SAC), and proximal policy optimisation (PPO). The policy is updated based on observations and rewards acquired from the environment in order to maximise the projected long-term reward. Learning with deep reinforcement learning algorithms is a dynamic process that occurs when the agent interacts with its environment. Deep RL algorithms makes it possible to learn a high-dimensional state space by using a neural network for functional approximation and thus stabilize the learning process.

DDPG refers to Deep Deterministic Policy Gradient, a deep reinforcement learning algorithm for continuous action spaces. It is an extension of the DPG (Deterministic Policy Gradient) algorithm that leverages deep neural networks to approximate both the policy and value functions. DDPG is particularly well-suited for problems where the action space is continuous, such as robotic control, autonomous vehicles, and other real-world applications. DDPG combines ideas from policy gradient methods and Q-learning. It uses the deterministic policy gradient to update the actor

network in the direction of higher expected cumulative rewards and uses Q-learning to update the critic network based on the Bellman equation. DDPG employs an actor-critic architecture, where actor network is responsible for approximating the policy function. while critic network is used to estimate the value function. Given a state-action pair, it predicts the expected cumulative reward starting from that state-action and following the current policy.

Soft Actor-Critic(SAC) is an advanced deep reinforcement learning algorithm designed for continuous action spaces. It is an extension of the DDPG and DPG algorithms, combining the strengths of both policy gradient methods and Q-learning. Similar to DDPG, SAC also employs an actor-critic architecture. It consists of an actor network that approximates the policy (the actor), and a critic network that approximates the action-value function (the critic). Unlike deterministic policy gradient methods, SAC uses stochastic policies, i.e., the actor network outputs a probability distribution over actions instead of deterministic actions. It uses the maximum entropy framework to encourage exploration and maintain a balance between exploration and exploitation. In SAC, entropy is maximized as part of the objective function during policy updates. This encourages the policy to be more exploratory, leading to more robust learning and better generalization to different environments. SAC also introduces a value function regularization term in the loss function. This allows the critic network to approximate the value function more accurately, leading to more stable and efficient learning. SAC is known for its effectiveness in solving complex continuous control tasks, especially in environments with high-dimensional action spaces, such as robotic control, autonomous vehicles, and many other real-world scenarios.

2.3 Multi Agent Reinforcement Learning

Multi-Agent Systems (MAS) [29] are systems in which multiple autonomous agents interact with each other to achieve individual and/or collective goals. Each agent in

the system is an autonomous entity capable of perceiving its environment, making decisions, and taking actions to achieve its objectives. In a multi-agent system, agents can be simple or complex, ranging from software agents running algorithms to physical robots or even humans in certain scenarios. These agents can have different capabilities, knowledge, and goals, and they often operate in a dynamic and uncertain environment. Multi agent systems have been found to be effective in different domains. In robotics, multiple autonomous robots can collaborate to accomplish tasks such as search and rescue, exploration, or warehouse logistics. Multi-agent systems can be used to optimize traffic flow and manage autonomous vehicles in smart cities. Similarly, Social networks can be considered as multi-agent systems, where users (agents) interact with each other to exchange information and influence each other's behavior.

Multi Agent Reinforcement Learning (MARL) is the application of Reinforcement Learning to multi-agent environments, in which various agents each have their own state-action space and share a common environment while still receiving rewards tailored to their own needs.

Formally, $MARL$ is defined as a tuple $(N, S, A_1, \ldots, A_n, p, R_1, \ldots, R_n, \gamma)$, where $N = 1, \ldots, n$ is the set of n agents, S denotes the set of environment states, $A_i, i = 1, \ldots, n$ denotes the finite set of actions available for the agents, $p : S \times A \times S \to [0, 1]$ denoting the transition probability for the states, $R_i : S \times A \times S \to [0, 1], i = 1, \ldots, n$ represting the rewards for the agents and γ signifying discount factor. At each state, the agent executes the action and the joint actions of all the agents determines the next environment state. Thus, at each time step t, agent $i \in x$, takes the action a_i^t with the current system state s_t. The agent then receives the reward $r_i(s_t, a_t, r_t)$ and the environment transitions to next state s_{t+1}. The agents then observe their own individual rewards. Agents in MARL settings can be cooperative or adverserial, depending on whether they are attempting to cooperate with one another to achieve a goal or acting against one another to maximise solely their own goal. The objective of the agent i is to optimize it's own reward according to the policy $\pi_i : S \to \delta(A_i)$,

such that $a_i^t \sim \pi_i(.|s_t)$. The state value function for the agent i, $v_i : S \to \mathbb{R}$, then utilizes the joint policy $\Pi : S \to \delta(A)$, where $\pi(a|s) := \prod_{i \in N} \pi_i(a_i|s_i)$. Thus, the expected discounted reward for each agent under the joint policy π, which assign the policy π_i for each agent i is,

$$v_k^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=0}^{T} \gamma^t r_k(t+1)|s_0 = s], \tag{2.1}$$

As a result, in MARL, the agent's policy is based not just on its own action but also on the actions of the other agents.

The approach of learning for the many agents in the MARL can vary, for example, independent learning, decentralised learning, centralized learning with decentralised execution (CTDE), and so on. Independent learning refers to a scenario where multiple agents learn individually without direct communication or coordination with each other. Each agent learns from its own experiences and data independently, without sharing information with other agents during the learning process. In this setting, each agent has its own model, and the learning algorithm is designed to optimize its performance based on its local data. The agents typically do not collaborate or share their learned knowledge, and they make decisions based solely on their individual experiences. Independent learning is common in scenarios where agents operate in isolated environments or have private datasets that cannot be shared due to privacy concerns. Examples include personalization in recommendation systems, individualized learning in education, and training personal AI assistants.

Decentralized learning, on the other hand, involves multiple agents or learners that communicate and collaborate during the learning process, but without a central coordinator. In this setting, agents exchange information, share knowledge, and learn from each other to collectively improve their performance. Each agent may have access to its local data, but the training process involves aggregating information from multiple agents to benefit from the collective knowledge of the group. The learning algorithms are designed to take into account information from different agents,

and the agents collaborate to find global solutions that improve the overall system's performance. Decentralized learning is useful in scenarios where agents can communicate and collaborate, but a central authority for coordination may not be feasible or desirable. Examples include collaborative multi-robot systems, distributed optimization in edge computing, and federated learning in a privacy-preserving setting.

Centralized learning with decentralized execution is a hybrid approach that combines the benefits of centralized learning and decentralized execution in the context of multi-agent systems or distributed computing environments. In this approach, the learning process is centralized, where a central authority or server collects data from multiple agents, trains a global model, and distributes the updated model back to the agents. However, the execution of the learned model occurs in a decentralized manner, with individual agents making decisions autonomously based on the shared knowledge. The primary advantage of CTDE is that it enables agents to provide and share more information about the state while training by aggregating their individual observations and experiences through centralised learning. As a result, agents learn to produce coordinated actions from partially observable individual states during the assessment step. This solution enables agents to use the observations of other agents to generate coordinated actions using a centralised critic framework that has access to the entire system's state as well as the individual agent's activities during the training phase. The critic module can examine how the joint policy affects each agent's long-term reward because it has access to all other agents' behaviours. As a result of sufficient observation and experience, the multi agent module learns to adopt coordinated behaviour.

2.4 Ranking metrics

Ranking metrics are evaluation metrics used to assess the performance of ranking models or any other algorithms that involve ranking items based on their relevance to a query. These metrics quantify how well the ranked list of items matches the

ground truth rankings provided in the evaluation data. The evaluation data typically consists of query-item pairs along with their true relevance labels. Building machine learning or deep learning models is based on the constructive feedback process. We create a model, get input from metrics, update it, and repeat it until we obtain the desired classification accuracy. The model's performance is explained by evaluation measures. The ability of evaluation metrics to discern between model outputs is a crucial feature. It is critical to understand the metrics used in machine learning (ML) systems. Ranking is a fundamental task in machine learning, recommender systems and information retrieval. We look at the most prevalent rank-aware measures for evaluating ranking tasks such as in learning to rank and recommendation systems:

The following rank aware metrics have been effective for evaluation :

- Precision, Mean Average Precision and Recall: Prediction accuracy and decision support measures fall inadequate when dealing with ranking tasks. Precision, recall, and F1 score are decision support metrics. These assess how well a recommender aids users in making sound decisions. They assist the consumer in choosing "good" products and avoiding "bad" items. These metrics are used to evaluate the effectiveness of ranking algorithms and information retrieval systems, especially when dealing with ranked lists of items. For instance, if we recommend 100 items to a user, the items in the first 5, 10, or 20 spots are most important.

  Precision is the percentage of selected components that are relevant to the user. It's primary goal is to recommend largely valuable content. The percentage of relevant items chosen by the system is referred to as recall. Precision and recall are frequently supplied with a top-n bound to broaden these measurements through metrics Precision@N and Recall@N provide this. Precision@N measure represents the percentage of "top-n" things that are relevant for the query/user and similarly with Recall@N. The metrics are calculated as , $Precision = \frac{RR}{TR}$; $Recall = \frac{RR}{TRD}$; , where RR refers to total relevant items

that are retrieved, TR refers to the total retrieved documents, TRD refers to total relevant items Search engines and recommender systems need to be able to put relevant items very high in the selected list.

Mean Average Precision(MAP) : Mean Average Precision metric is extended from precision and also used to evaluate the performance of ranking models for document/information retrival. The Average Precision(AP) measure indicates how much of the relevant documents are contained in the top-ranked documents, AP = $\sum_K (Recall@k - Recall@k\text{-}1)$. This metric automatically manages the ranking of recommended items in lists. Again mean of AP over all the queries delievers the mean average precision. This is in contrast to metrics that treat retrieved items as sets. This statistic can assign extra weight to errors that occur near the top of the suggested lists. In contrast, it pays less weight to faults that occur further down in the recommended lists.

- NDCG: Normalized Discounted Cumulative Gain The NDCG is metric of a ranking system's efficacy that takes into account the location of relevant items in the ranked list.. The MAP metric's goal is similar to the NDCG metric's goal as both place a strong importance on putting highly relevant documents at the top of the result list returned. The NDCG fine tunes the examination of recommended lists and can make use of the fact that certain documents are "more" relevant than others. Highly relevant items should be prioritised above medium relevant ones, which should be prioritised over least relevant items or documents. Cumulative gain is a fundamental measure for accumulating graded relevances however it does not consider the order of the elements in the ranked list. We need to increase the relative importance of the position of components in the ranked list for ranking assignments. The typical Discounted Cumulative Gain, DCG, incorporates a logarithmic reduction factor to penalise the relevance score proportionally to the item's location. DCG either increases with list size or remains constant. This means that queries with large result sets will almost always have higher DCG ratings than searches

with relatively small result sets.To make comparisons between inquiries more equitable, normalise the DCG score by the maximum attainable DCG at each threshold k. The NDCG is given as, $NDCG\ @k = \frac{DCG\ @k}{IDCG\ @k}$, where k is size of the list and $IDCG\ @k = \sum_{i=1}^{k} \frac{2^{rel_i}-1}{log_2(i+1)}$, where $IDCG$ represents the ideal DCG that can be obtained by sorting all the items in the list by their relevance.