SOCIAL REINFORCEMENT LEARNING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Mahak Goindani

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2020

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Jennifer Neville, Chair

Department of Computer Science

Dr. Clifton Bingham

Department of Computer Science

Dr. Dan Goldwasser

Department of Computer Science

Dr. Petros Drineas

Department of Computer Science

Approved by:

Dr. Kihong Park

Head of the School Graduate Program

To, who have always stayed

through my thick and thin:

The Glorified Almighty,

My Adored Parents,

My Revered GrandParents,

My Beloved Brother,

My Dearest Uncles

ACKNOWLEDGMENTS

I wholeheartedly thank my advisor Dr. Jennifer Neville, who has been a great mentor and has played a significant role in the successful completion of this dissertation. Jen is one of the most empathetic people I have met in my life. I admire her for her compassion and humility, and I am inspired by her professionalism, hardwork, calm and positive attitude even during tough times. I am grateful to her for providing timely feedback on my progress and guiding me through every step in research, starting from literature survey to problem formulation to designing solutions and experiments for evaluation. I thank her for being patient through these, and providing me the freedom to explore different research topics, problems and solutions. Her technical expertise and abundant knowledge on a variety of areas have helped me improvise my solutions, and thanks to her wealthy experience, sharp insights and unique perspective that played a key role in the successful completion of this dissertation. I understood the true meaning of research after I started working with her. I also thank her for teaching me the art of writing good papers and presentation of ideas with clarity. I learned from her to constructively deal with criticism and use it to improve solutions and presentations in the future. I was fortunate to work both as a Teaching Assistant and a Research Assistant under Jen. The learnings that I received from Jen as a TA have helped me improvise my communication and presentation skills, in my own research and professional career. I also admire her for taking interest in students' concerns and going above and beyond to help them resolve those, be it related to research, or internships or jobs, or otherwise. I also appreciate her efforts to involve some humor during our meetings, that helps to reduce stress, and easy conveyance of ideas. I also thank her for providing me valuable tools and resources to accomplish my research and teaching activities. I will always be obliged to Jen for her wonderful guidance and incessant support and encouragement, even amidst

a global Covid-19 Pandemic and on her sabbatical, that made my Ph.D. journey a great learning experience. I am grateful to her for mentoring me on both professional and personal fronts that has helped me grow as a better individual. I will always cherish the meetings and interactions with Jen.

I am obliged to my committee members Dr. Chris Clifton, Dr. Petros Drineas, and Dr. Dan Goldwasser for their valuable feedback and thoughtful comments on my dissertation. I benefited enormously from the insightful discussions with them.

I have been greatly fortunate to have an immensely loving, caring and supportive family members, who have always made sure that I am happy. I will be forever indebted to them for all the things they have done for me, and have raised me as a better individual. I am grateful to my parents Dr. Suresh Goindani and Dr. Renu Goindani, and my grandmother Lajwanti Dembla, for bestowing their infinite blessings and unconditional love upon me, and for their endless sacrifices to ensure that I achieve great success in life. It was my parents' motivation that helped me acknowledge the importance of research and education in life. They have always strived hard to make sure that I receive the best education and resources in my life. I will always be thankful for their incessant efforts and countless nights they have spent to guide and encourage me to excel and reach the zenith. I am thankful to their unconditional moral support, even from great distance and different time zone in India. They are my back bone, and it is their guidance, love, support and understanding, that has helped me survive through all ups and downs in my life and research. I thank them for being available always and listening to my anxiety and frustrations, and sharing their experiences and advice to boost self-confidence in me. This has helped me avoid any distractions, and focus solely on the objectives with full concentration and dedication. I thank them for always guiding me to stay calm, composed and positive in all circumstances in life, which has helped me focus on improving myself and provided strength to face criticism. They always advised to not think deeply about failure, but rather learn from it and move faster and stronger towards my goals. They have always ensured to ask me about my progress at school and my work, which helped me move

forward constantly without getting deviated from my path. I am grateful to their constant efforts for always being available and making me feel better — away from home. They have always motivated me to think big and put in great efforts to achieve heights in my career. They are my role models and I admire them for their hard work, dedication, and sincerity. They have inculcated self-discipline in me, which is a key factor for successfully completing my dissertation. I faithfully thank Mr. Narendra Dubey for his encouragement and teachings to always be positive and strive towards my goals. His precious blessings and constant moral support have helped me come this far. I express my sincere gratitude towards my grandmother, who has taken a great care of me and bestowed her selfless love, due to which I could achieve success. I also thank my brother Akshay for all his help, prayers, and the great discussions we have. His lively, humorous, and joyful talks always helped to alleviate the stress during my Ph.D., and are a reason for my happiness. I am greatly thankful to my uncles Mr. Naresh Dembla, and Mr. Navin Dembla, for their unceasing wonderful guidance, advice and support to overcome different hurdles in my journey. They have always taught me the importance of education. They have shared valuable resource to enhance the quality of my work, and have always motivated me to study further. I thank Uncle Naresh for his constant help and motivation to improve my skills and knowledge while in India. He constantly reminded me of my objectives and guided me through to complete my dissertation faster. Uncle Navin has always provided me a family-like home in the US, and I thank him for his constant efforts to make sure that I stay happily and comfortably away from home. I thank them for motivating me to go beyond my capacity to achieve greater. I will always be obliged to my friend Louise Jewell, who has helped me throughout my stay in West Lafayette. I thank her for her great companionship and welcoming nature for international students, along with the numerous rides, good food, and other resources that she gave me, to make my stay easier in this place, away from my family. I am falling short of words to express my gratitude and praise for my family members for all their selfless sacrifices, efforts, love, support and encouragement, without which this journey would have been

greatly challenging, especially at a physical distance from family members. I couldn't have asked for anything more in life.

I express my heartfelt gratitude for my colleagues in the Network Learning and Discovery (NLD) Lab under Jen, for their constructive comments and helpful discussions on my research. Hogun has always been encouraging and supportive, and I thank him for listening to my concerns and sharing advice with me, that helped me bear the stress during challenging times. I am very much grateful to Gui and Jean for taking out their worthy time to discuss my ideas and provide constructive feedback to improvise solutions. Their keen insights have helped me make my approach more methodical. I am also thankful to Ellen, Gui, Jean, Mengyue, and Giselle for their valuable suggestions to improve my presentations, and for helping me realize the importance of great presentations. I am grateful to Jason and Jiasen for sharing great advice and tips to improve my dissertation. I also thank Jean and Mengyue for their friendly conversations and generous rides. Thanks to the kind help from my amiable labmates that this journey became joyful and easier. I also thank my other peers at Purdue for helping me through difficult times. Ahmed M., and Ahmed E. have always motivating me to achieve greater by sharing their experiences, that helped me become more confident, especially in trying times. I am also obliged to the Computer Science Department at Purdue, and the staff members, for providing me funding, resources and a welcoming environment to pursue research in the United States. I also thank all professors at my undergraduate school, who gave me their valuable recommendations, which also played a role in my successful admission to the doctoral program.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Goindani, Mahak PhD, Purdue University, December 2020. Social Reinforcement Learning. Major Professor: Jennifer Neville.

There are various real-world applications that involve large number of interacting agents, for e.g., viral marketing, personalized teaching, healthcare, recommendation systems, online communication platforms. However, much of the existing work in Multi-Agent Reinforcement Learning (MARL) focuses on small number of agents. The standard approaches to train a complex model for each user in a decentralized fashion are impractical for thousands of agents. Centralized learning is also infeasible due to the curse of dimensionality and exponential increase in joint representations. There is an opportunity to utilize the interactions and correlations between agents, to develop RL approaches that can scale for large number of agents. However, user interactions are typically sparse. In this dissertation, we define *Social Reinforcement Learning* as a sub-class of MARL for domains with large number of agents with relatively few (sparse) relations and interactions between them.

We consider the important task of fake news mitigation as an example to demonstrate the real-world applicability of our proposed Social RL approaches. First, we propose a centralized Social RL approach to estimate incentives (interventions) required to promote the spread of true news in a social network—in order to mitigate the impact of fake news. We model news diffusion as a Multivariate Hawkes Process (MHP) and make interventions that are learnt via policy optimization in a Markov Decision Process (MDP). The key insight is to estimate the response a user will get from the social network upon sharing a post, as it indicates her impact on news diffusion, and will thus help in efficient allocation of incentive. Second, we develop an efficient centralized Social RL approach to address the challenges of computational

complexity (associated with large number of agents), and sparse interaction data. Our key idea is to reduce the model size by *dynamically* clustering users based on their *payoff* and *contribution* to the goal. Lastly, the above proposed centralized approaches can be applied when the environment is fully observable to all agents, with a common system shared between all agents. To develop solutions for scenarios where agents receive only a partial view of the environment, and agents can also have separate individual goals, we propose a Social RL approach that is more decentralized. Our key idea is to use sequential parameter sharing and *ego-network extrapolation* to incorporate agent correlations and improvise estimates of the partially hidden system information. We evaluate our proposed approaches on two Twitter datasets. Our centralized learning methods outperform other alternatives that do not consider estimates of user feedback when learning how to allocate incentives. Furthermore, by clustering users, we are able to achieve faster convergence along with learning more accurate estimates, compared to baselines that do not model agent correlations or only use static clusters. Additionally, our decentralized learning approach achieves performance equivalent to that of centralized learning approach and superior performance to other baselines that either consider complete system information available to an agent, or other estimates of the hidden environment state.

1 INTRODUCTION

With the increased inclusion of social network interactions in media, information, and online retail systems, there is an opportunity to learn from relational interactions and user feedback to improve overall system engagement and better personalize customer recommendations. However, since user behavior evolves over time and is influenced by both peers and changes in the environment, it is essential to tailor network strategies by learning sequential decisions *interdependently* and *dynamically*.

Reinforcement Learning (RL) is useful to model scenarios when *agents* interact with the environment leading to a change in their behavior or *state*. Additionally, there can be multiple agents interacting with each other and with the environment, that again influences their decisions or *actions*, and this comes under the class of Multi-Agent Reinforcement Learning (MARL). As opposed to Supervised Learning, there is no direct signal (labeled input/output) to learn from in RL. The signal in RL is the dynamic *reward* obtained on interactions with the environment. The objective of RL/MARL is to learn a *policy* function that helps the agent determine the best action to take, in a given state, in order to maximize the reward.

In this thesis, we propose *Social Reinforcement Learning* as a general framework to facilitate modeling in a dynamic environment with user interactions and feedback. The setting is applicable for a variety of real-world network applications that have (i) large number of users taking actions, (ii) user-user or user-item interactions, and (iii) inter-dependence between user actions, i.e., actions taken by one user influence the actions of other users in the network. This includes domains such as viral marketing, personalized teaching, healthcare, and recommender systems. Specifically, we define a Social RL model for learning in scenarios with multiple interacting agents—to learn decisions that help to maximize the likelihood of desired consequences in the future. We note this is different from existing work in MARL as we consider a large

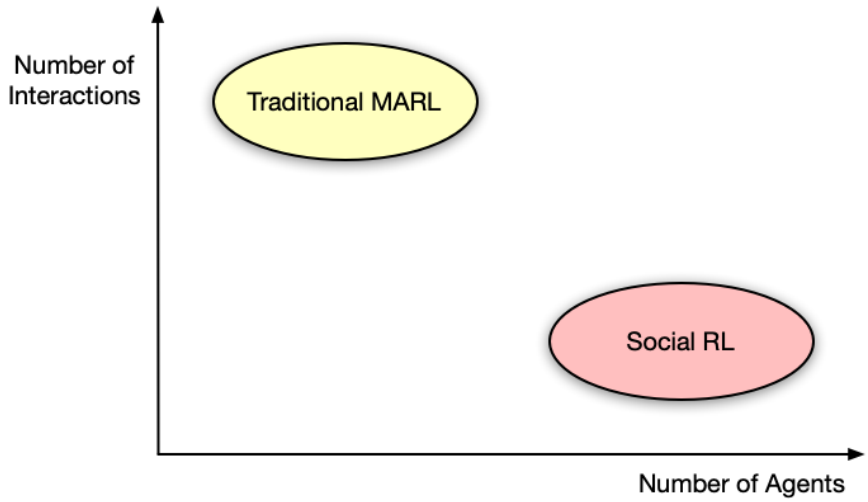


Figure 1.1.: Traditional MARL vs Social RL

number of agents with relatively sparse interactions between them (Fig. 1.1), which make it challenging to learn inter-agent *dependencies*. In real-world social networks, user interactions lead to dependencies between their actions (due to peer-influence). Specifically, the actions taken for one user impacts the likelihood of the actions for other users in the future. For example, if one user tweets more about a certain topic, that may influence their followers to tweet more on the same topic. Thus, the objective is to learn strategies that scale for large number of agents, and take the inter-agent interactions into account. However, we note that each user typically only interacts with a small number (i.e., $\ll N$) of other users in the network. This means the network interactions are relatively sparse.

The goal in this setting is to learn a policy π that maps user(s') *state* to user(s') *actions* — to determine the best action(s) to take to maximize *reward* (which is determined by transitions to future states). For example, we may want to learn how to promote the release of new movies to users in a social network, to encourage liking/reviewing/sharing activities with their friends, in order to maximize ticket/rental purchases over time.

1.1 Application

There are several critical real-world applications where the Social RL setting is applicable, e.g., viral marketing, healthcare, recommender systems with community interactions. Social RL helps to model system dynamics such as estimating user responses (e.g., likes, comments, shares) considering effects of peer-influence. It can be used to understand customer demands, and learn strategies to incentivize competing products (e.g., news recommendation) by capturing changing user interests and feedback. Thus, it is essential to develop MARL solutions that scale for large number of agents, for challenging dynamic environment comprising of high-dimensional observation spaces, in collaborative and adversarial settings, with partial observability.

In this dissertation, we consider an application of Fake News Mitigation, and develop Social RL approaches to combat fake news spread in social networks. [1] defines fake news as "fabricated articles that are intentionally and verifiably false, and could misled readers". Online social media has increasingly become a platform for widespread dissemination of fake news stories [2]. [1] found that roughly half of the users on Facebook who viewed fake news stories believed them. The diffusion of fake news in social networks is significantly farther, faster, deeper, and wider than that for true news [3]. This indicates a pressing need to combat fake news spread in social networks.

Fake news mitigation is a multifaceted problem. Much of the previous work has focused on the detection of fake news using linguistic, demographic, and community based features. [4] studied network properties such as clustering coefficient, closeness and betweenness centrality, neighbor based features like number of followers and followees, to identify users likely to spread fake news on Twitter. [5] tried to classify the propagation path of news to detect fake news at early stages of diffusion. There has been relatively less work on limiting the *spread* of fake news. Some recent work has considered mitigating fake news by identifying potential purveyors of fake news to block their posts [6, 7]. However, it may not be feasible to take forceful actions

such as censoring users posts, since it can violate users' rights (Bill [8]), and can have negative effects enforcing misinformation [2, 9–11]. Some research tried to limit the influence of fake news by strategically selecting users that can spread true news ([12–14]). We use an approach similar to [15], which aimed to mitigate the impact of fake news by making interventions to the *true* news diffusion process. In addition, we also consider the user response as *feedback* to determine the efficacy of users, and model both the news diffusion and user responses as stochastic processes. [16] observed that around 46% of the fake news stories circulated on Facebook were on U.S. Politics and Elections. In addition, the reactions of people are more significant for topics related to politics, than other topics such as movies or weather [17]. Thus, in this work, we consider news related to U.S. Politics.

The problem of fake news mitigation can be mapped to a Social RL setting, as social networks comprise of thousands of agents who interact with each other by sharing information. There are community interactions on social media where users share news articles corresponding to fake or true news, and other peers provide comments or likes. However, typically social network interactions are sparse as users interact with a relatively small number of other users. To mitigate the impact of fake news, we can incentivize users to spread more true news, by learning interventions for the true news diffusion process, such that the users who are exposed more to fake news also become exposed more to true news. We provide the detailed problem definition in Sec. 5.3.

1.2 Research Questions

Social reinforcement refers to the process where acceptance and praise from others reinforces behaviors/preferences of an individual (e.g., [18]). With the increased use of social media, response from peers plays an important role in shaping the behavior and decision-making of individuals (e.g., [19]). In this research, we define a Social Reinforcement Learning Problem that considers learning in environments with mul-

tiple interacting agents—to facilitate modeling user interactions and feedback while learning decisions (actions) for a given situation (state) that maximize the likelihood of desired *reward* in the future. Additionally, the environment can be fully observable, where all agents observe the complete network state, or it can be partially observable, where an agent receives a partial view (observation) of the network, that may be different from other agents' observations. Agents can receive a shared network reward based on the collective actions of all agents, or they can receive local rewards based on their individual actions. Social RL is different from traditional MARL as it considers a large number of agents with sparse interactions between them. Agents interact with each other that that leads to an increase in the likelihood of certain actions in the future due to peer-influence and effect of past interactions, and this impact decays over time. Due to this, there is a temporal dependency between agents' states and actions, and hence, the data is not independent and identically distributed, as is in many other problems. Thus, we develop solutions that can scale for large number of agents, and capture the relations and interactions between agents, along with the feedback received from the environment — to learn to make better decisions in both fully observed and partially observed scenarios where agents can receive global as well as local rewards.

1.2.1 Main Hypothesis

The goal of the present research is to verify the following hypothesis. Social Reinforcement Learning considers the setting with large number of agents with relatively few interactions between them. It is, thus, *challenging due to high-dimensional search space, and sparse agent interactions*, resulting in *high computational complexity, and insufficient samples to capture inter-agent dependencies for learning accurate policies*, especially in partially observable environment. By utilizing the properties of the *social network structure, agent relations and interactions*, we can obtain a *compact model* to represent the environment dynamics and estimate latent environment state.

This can help in *learning more accurate policy estimates*, along with achieving *faster convergence* because *agents are correlated (behave similarly) and peer influence and feedback in social networks helps determine user efficacy.*

1.2.2 Proposed Research

We propose the following solutions to overcome the challenges in Social RL problems, and enhance policy learning by effectively using properties of the social network structure, and agent relations and interactions.

(a) We use Multivariate Hawkes Processes (MHP) to characterize user activities in social network. By *integrating the MHPs in a RL framework*, we model both excitation events and social reinforcement. Our key insight is to estimate the *response* a user will obtain from the social network upon sharing a post, and use it to learn better policies in such a way that we avoid full joint learning or learning an independent model for each user, that are both computationally intensive. User response helps in learning appropriate selection of users and efficient allocation of incentive among those, under budget constraint.

(b) Due to large number of users and high-dimensional continuous spaces, centralized learning becomes computationally intensive. To overcome this, we propose a *cluster-based policy learning* approach that utilizes the correlation between agents to reduce the dimensionality of state/action spaces along with avoiding noisy estimates. Our idea is to by *dynamically* cluster users (based on their *payoff* and *contribution* to the goal) and combine this with a method to derive personalized agent-level policies from cluster-level policies. This helps to reduce the effective number of agents and offset sparse interactions, thus achieving faster convergence and better policy estimates.

(c) Dependencies among user activities throughout the network impact the reward for individual actions and need to be incorporated into policy learning, how-

ever the directed interactions entail that the network is partially observable to each user. To address this, we consider decentralized learning and execution, in contrast to the above solutions that use centralized learning and execution. Since it is challenging for users to capture network dependencies while learning policies *locally*, due to insufficient state information, we propose to use parameter sharing and *ego-network extrapolation* to incorporate agent correlations and improve estimates of the partially hidden state information, for learning better policies.

This document is organized as follows. First, we provide background on multi-agent reinforcement learning and multivariate Hawkes processes. Second, we define Social Reinforcement Learning, and describe the challenges and opportunities to solve Social RL problems, along with the real-world applications where Social RL setting is applicable. Third, we describe related work with respect to the proposed problems. Fourth, we present a centralized Social RL approach that uses feedback modeled as a function of peer-influence and political bias, to obtain an effective incentive allocation strategy under fixed budget. Fifth, we describe an efficient centralized cluster-based social reinforcement learning approach that utilizes agent correlations for learning better policy estimates and reducing the computational cost. Lastly, we introduce partially observable setting in Social RL, along with local rewards, and propose solutions for learning individual policies more accurately and efficiently, by modeling dependencies in ego-networks, and extrapolating those.

2 BACKGROUND

Reinforcement Learning (RL) is used to solve sequential decision making problems in which agents learn by interacting with the environment. Agents perform *actions* that lead to a change in their *state*, and receive *reward* based on their actions. This reward is used as signal by the agent to learn optimal actions to perform in the future. On performing an action, the agent transitions to another state with some probability. Thus, the environment is stochastic in nature as there is an uncertainty about the consequences of actions, due to which the next state of the environment is not certain. The next state (input) of the environment depends on the current state and/or previous states, and thus, the input is not independent and identically distributed (non-i.i.d).

This is different from supervised learning or semi-supervised learning problems that consider i.i.d. data with pre-specified labels provided for learning a mapping from inputs to outputs. In RL, there are no pre-specified labels given, instead, the agent learns using the reward obtained by performing actions. In unsupervised learning, there are no labels provided for the data. Thus, RL is also different from unsupervised learning since the dynamic reward is used as the signal for learning.

2.1 Markov Decision Process

Markov Decision Processes (MDP) are used to solve RL problems. A MDP is a tuple $\langle \mathbf{S}, \mathbf{A}, R, T, \gamma \rangle$, where $\mathbf{S} = \{s_t\}_{t=1}^{T}$ is the set of states at different time-steps t, $A = \{a_t\}_{t=1}^{T}$ is the set of actions, $R(s_t, a_t) \in \mathbb{R}$ is the reward obtained on taking action a_t in state s_t, $T(s_t, a_t, s_{t+1})$ is the probability of transitioning from state s_t to the next state s_{t+1} on taking action a_t, and $\gamma \in [0, 1]$ is the discount factor that governs the relative importance of immediate and future rewards. The objective is to learn a

policy function $\pi : \mathbf{S} \rightarrow \mathbf{A}$ that maximizes the total expected discounted reward for all stages $\sum_{t=1}^{T} \gamma^{t-1}\mathbb{E}[R(s_t, a_t)]$. Each state is associated with a value function $V_\pi(s_t)$, that is the total expected reward when in the given state s_t following policy π, i.e., $V_\pi(s_t) = \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1}R(s_t, \pi(s_t))]$. Each state-action pair is associated with a Quality function or Q-function, that is the total expected reward obtained on taking action a_t in state s_t and then following policy π, $Q_\pi(s_t, a_t) = \mathbb{E}[R(s_t, a_t)] + \sum_{t=2}^{T} \gamma^{t-1}\mathbb{E}[R(s_t, \pi(s_t))]$. In multi-agent RL, agents can either receive individual rewards R_i or a common reward R shared between all agents.

MDPs use the Markov property. Specifically, we assume that the current state alone affects the next state, i.e., the future (s_{t+1}) is conditionally independent of the past $(s_1, s_2, ..., s_{t-1})$ given the present state (s_t). This assumption requires the states to be fully observable to each agent. A generalization of MDP is Partially Observable MDP (POMDP) that considers the scenario when agents receive a partial observation of the system $o_{t,i}$, and each agent takes action based on her local observation.

2.2 Policy Learning and Optimization

The state-of-the-art Deep RL methods use *Policy Search* to learn the optimal policy function. Specifically, a policy is approximated as a function parameterized by weights θ, i.e., $\pi = f(s_t; \theta)$, and the parameters are updated in order to maximize the expected return. Deep Neural Networks are widely used to approximate the policy function as they are powerful function approximators that can effectively capture higher-order relations between different states and actions [20, 21]. Policy Gradient methods are gradient-based optimization methods used to learn optimal parameters, and are more effective in high dimensional spaces, and can learn continuous policies [22–25]. The gradient of policy estimates provides strong signals to improve the parameterized policy, however, these estimates have high variance, especially for high-dimensional state-action spaces [26].

To reduce the variance, a widely used approach is to subtract a baseline from the objective function to get unbiased estimates that are less noisy [27]. Instead of simply using the expected reward, an advantage function is to optimize the policy that is obtained by subtracting the value of state as baseline from the expected reward. This is known as the Advantage Actor-Critic algorithm [25, 28, 29], where the policy function (actor) learns using the feedback from the value function (critic). Intuitively, the advantage function helps to determine the action that has better consequences compared to others (relative advantage), by removing an average amount of return. Since it is computationally expensive to compute $V(s_t)$ from future rewards for every possible policy π, the value of a state is approximated as a function parameterized by weights ϕ, that is, $V(s_t) = f(s_t; \phi)$ (e.g., [30, 31]). Generally, for stable learning, separate parameters are used to approximate the policy function and the value function.

2.3 Reward Shaping

Learning the policy based on the global system reward, without considering the individual contribution of users results in noisy estimates, as an agent may be rewarded positively for performing bad if other agents are performing good [32]. The idea behind *reward shaping* is to shape the reward signal so that it better reflects an agent's contribution, to improvise MARL solutions. Difference Reward (DR) and Potential-Based Reward Shaping (PBRS) are two widely used reward shaping techniques.

Difference Reward (D_i) is a shaped reward signal that helps to determine the effectiveness of an agent by subtracting the contribution of other agents from the overall system objective, i.e., $R_i = R - R_{-i}$, where R is the global system reward, and R_{-i} is the system reward without the contribution of agent i. This also helps to remove a large amount of noise created by the actions of other agent in the system [33, 34]. Any action taken to increase R_i corresponds to increasing R, while agent i's impact

on her own reward is much higher than its relative impact on R [35]. Thus, DR is used to enhance policy learning in multi-agent systems.

In Potential-Based Reward Shaping, the shaped reward signal is the the difference of a potential function defined over a source state and a destination state [36]. However, PBRS requires prior knowledge of the problem domain and the potential function is application-specific. Hence, PBRS has limited utility. [35] proposed combination of DR and PBRS to improve the reward signal for policy learning.

2.4 Multivariate Hawkes Process

Hawkes process [37] is a stochastic point process governing the arrival rate λ of an event. It is self-exciting in nature, which means that past events increase the likelihood of occurrence of new events in the future. Let $\mathcal{N}(t) \in (\mathbb{Z} \cup \{0\})$ be a counting process representing the number of events upto time t. Let t_i be the time of occurrence of i-th event. The Hawkes process is represented by the counting process,

$$\mathcal{N}(t) = \sum_{t_i \leq t} h(t - t_i)$$

where $h(t) = 1$ if $t \geq 0$, and 0 otherwise. Let $\lambda(t)$ be the associated intensity function governing the arrival rate of events. For Hawkes process, $\lambda(t)$ is given as,

$$\lambda(t) = \mu + \sum_{t_i \leq t} \mho(t - t_i) \tag{2.1}$$

where $\mu \in \mathbb{R}^+$ is the base exogenous intensity and $\mho(t)$ is the Hawkes kernel that determines the decay rate of the influence of past events. For example, the Hawkes exponential kernel is given as $\mho(t) = \phi\, \omega e^{\omega t} h(t)$, where $\phi > 0$.

N-dimensional Multivariate (multi-dimensional) Hawkes Process (MHP), consists of N Hawkes processes $\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_N$, that are mutually-exciting in nature, that is, the occurrence of an event of process (dimension) i influences the occurrence of an event of process (dimension) j in the future. In this case, each process (dimension) has its own intensity function, given as,

$$\lambda_i(t) = \mu_i + \sum_{j=1}^{N} \int_0^t \Phi_{ij} \omega e^{-\omega t} d\mathcal{N}(s) \tag{2.2}$$

where s is the placeholder for limits in the integral, and Φ_{ij} is the influence that process i exerts on process j. MHPs have been widely used to model user activities in social networks [15, 38].

2.5 Political Bias

We consider news related to U.S. Politics (Chapter 1), and measure *political bias* of a user as the political leaning of the user towards communities of two polarities: Democratic (D) and Republican (R) Party. Each user i has bias values $b_{R,i}, b_{D,i} \in [0, 1]$, for R and D, respectively. To compute the initial bias values, we run a random walk based community-detection algorithm [39, 40], using the adjacency matrix of the social network, with starting seeds for the two communities as the official profiles of politicians whose political affiliation is already known. The bias is estimated as user's proximity to the two sets of seeds for D and R such that $b_{R,i} + b_{D,i} = 1$.

Bounded Confidence Model

According to Bounded Confidence Model (BCM) [41, 42], each user i has an opinion $x_i \in [0, 1]$. Two adjacent users i and j interact if and only if their opinions are close enough, i.e., $|x_i - x_j| \leq \epsilon \in [0, 0.5]$, resulting in a change in their opinions as $x_i = x_i + \mu(x_j - x_i)$ and $x_j = x_j + \mu(x_i - x_j)$. BCM assumes that the two interacting users must be close enough in their opinions (hold same set of beliefs), and the exchange increases one user's opinion and decreases another's. However, these assumptions do not hold in our case, since an interaction between users with similar ideology can increase the bias instead of reducing it. Moreover, we have a one-way interaction between a user and her followers instead of both ways. Therefore, we proposed a model to update political bias (described in Chapter 6).

3 SOCIAL REINFORCEMENT LEARNING

3.1 Motivation

With the increased use of social media, ratings and response from peers plays an important role in shaping the behavior and decision-making of users (e.g., [19]). While there has been a great deal of research in the IR and RecSys communities on how to incorporate user activity and feedback to personalize search and recommendation, there has been relatively less work on how to model dependencies among dynamic user interactions in social networks for these same tasks. Real-world social networks consist of large numbers of users who interact with each other and are related in various ways ([15, 43, 44]). There is an opportunity to learn relationships between the users (i.e., agents) based on their network interactions over time and use the discovered correlations to improve automated decision making. For example, estimating user responses (or feedback) in social networks (likes/comments/shares) helps to learn strategies for motivating users to favor one competing product over another [43] to increase brand awareness and revenue, or deciding how to place content on websites to maximize click-through-rate (e.g., [45]) and seek customer attention. Also, retail websites can enhance their recommendations by utilizing the user-feedback via reviews/ratings on purchased products, to increase the overall number of returning visitors (described more in Sec. 3.3.3), or online question-answer forums (e.g., StackOverflow) can exploit user-interactions via answers/comments to other users' questions, to increase the rate of providing answers, or minimize the number of unanswered questions (Sec. 3.3.4). Furthermore, online communication platforms (e.g., Slack) can learn to identify *important* people for a particular user, at different times, based on feedback such as frequency of interactions or delay in responses, and prioritize messages from them, or suggest labels/filters for messages from different peers, in order to improvise the

quality of service. Email services can utilize peer communication patterns to learn auto-fill suggestions/salutations and personalize those for different peers with different relations (formal/informal contacts). In corporate organizations, a manager can learn which comments to make, when to comment, whom to reply to in an online group discussion (for a new feature/service) with her team, so as to encourage more people to participate or respond faster. [46, 47] showed that social networks play an important role in the development of obesity via peer influence on people's energy intake and physical activity. However, appropriately selecting users based on social network structure for anti-obesity campaigns can help to reduce individual's body weight and thus, overall obesity prevalence in the complete network [48]. Moreover, social media can be utilized as a promising intervention platform for increasing physical activity among people by providing supportive social influences [49, 50]. Thus, personalized healthcare applications can utilize peer-influence to encourage users to exercise more and lose weight in a shorter period of time (Sec. 3.3.5).

There are numerous settings that involve large numbers of interacting users, including social networks, online advertising bidding agents [51], healthcare, recommender systems with community interactions, online communication platforms and email services. It is essential to develop sequential decision making methods for these domains that scale to a large number of interacting users. Note that user preferences in these systems may evolve based on peer behavior or environment dynamics. In this case, a personalized recommendation systems (e.g., news) need to update and/or diversify its suggestions to keep up with dynamic user interests [31]. Moreover, there are also applications (e.g., viral marketing) where multiple users share a limited resource yet influence others in unobserved ways, which requires learning how to assign credit [35]. These characteristics lead to a dynamic environment comprising of high-dimensional observations, which further challenge the development of sequential decision making methods.

We consider a Multi-Agent Reinforcement Learning (MARL) based solution, which models multiple agents that interact with the environment and with each other, lead-

ing to a change in their states and actions in the future. The use of RL facilities incorporation of feedback in the form of immediate as well as future rewards to optimize decisions based on past experiences, and consider actions with long-term utility. This is particularly useful when the reward is delayed over time. For example, there are different applications that motivate users to follow a healthy lifestyle like exercising regularly by providing them small immediate incentives, and larger incentives in the future when they achieve a milestone after performing a sequence of actions.

3.2 Problem Definition

We define Social Reinforcement Learning as a sub-class of Multi-Agent RL, that considers large number of agents with interactions between them. Specifically, we consider a social network setting with N users. Each user $i \in \{1, ..., N\}$ is an agent. Let $G = (V, E)$ represent the social network graph, where each node $v_i \in V$ corresponds to user i, and edge $E_{ij} = 1$ if there exists an edge (i.e., relation) between agent i and j, and 0 otherwise. In addition, users perform d different activities (e.g., tweet, comment, like) in the social network. We say that the network is dense when all agents interact, i.e., the number of interactions is $\Omega(N^2)$, and the network is sparse when agents only interact with a constant number of other agents, i.e., $O(N)$ interactions.

Let $\mathbf{s}_i \in \mathbb{R}^d$ ($\mathbf{s}_i \geq 0$) be the state of user i. This corresponds to the d activities the user performs. Then, the state of the network represents the activities over the N users, $\mathbf{s} = \{\mathbf{s}_i\}_{i=1}^N$. The network activities are dynamic in nature, and thus, we have different states of the network at different time-stamps. Let $\mathbf{s}_t$ represent the network state at time t. We consider a finite horizon setting, i.e., we observe the activities up to time T.

An action $a_{d,i} \in \mathbb{R}$ corresponds to a modification to the d-th activity of user i. It is important to note that action is different from activity. An action is a decision to influence an activity or make the activity more visible. Let $\mathbf{a} = \{a_{d,i}\}_{i=1}^N$ refer to the set of actions, one for each user in the network. Actions leads to a change in the state

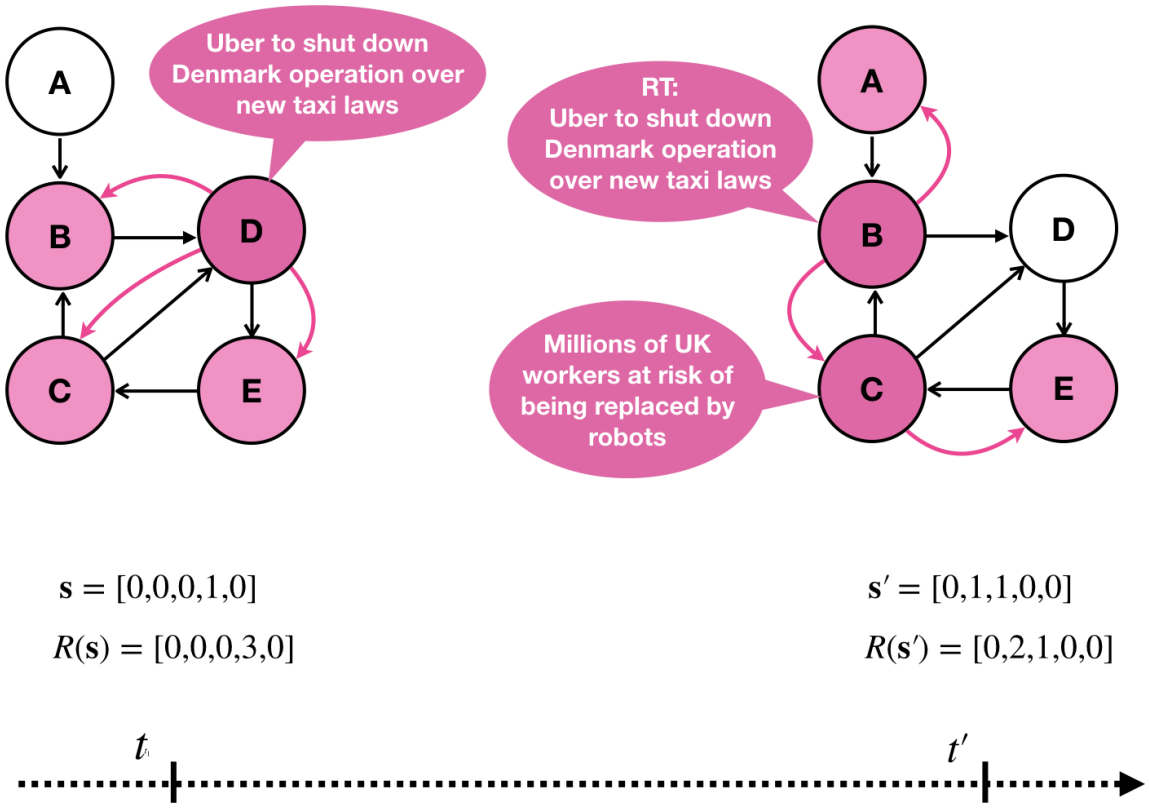


Figure 3.1.: Social network with agent interactions. Colored nodes represent active users (darker shade for users who are tweeting, lighter shade for people exposed to tweets). State is given by the number of tweets by each user (in the order [A,B,C,D,E]), and reward is calculated as the number of users exposed. Users' collective actions of tweeting news lead to a transition from state $\mathbf{s}$ to $\mathbf{s}'$.

of each user, and consequently, a change in the overall network state. Let $\mathcal{T}(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ be the probability of transitioning to the network state $\mathbf{s}'$ after performing actions $\mathbf{a}$ in network state $\mathbf{s}$. There may be a reward $R_{d,i}$ for each agent i and/or each activity d, or a collective reward $R(\mathbf{s}) \in \mathbb{R}$ based on the complete network state that is shared between all agents.

Because users interact in the network, actions taken for one user may impact the state (i.e., activity) of other users. More specifically due to peer-influence, changes in the activity of user i may influence the likelihood of user j's activities in the future

if i and j are connected in $\mathbf{G}$, either directly (i.e., $E_{ij} = 1$) or through a longer path. For example, if one user tweets more, that may influence their followers to tweet more as well. Fig. 3.1 shows a network of users who interact via sharing news.

Thus, agent interactions lead to dependencies between agents' activity, and thus actions and state transitions. Specifically, the state transition distribution does not depend on the user activities $\mathbf{s}$ and actions $\mathbf{a}$ alone, but also depends on the network $\mathbf{G}$, i.e., $P(\mathbf{s}'|\mathbf{s}, \mathbf{a}, \mathbf{G}) \neq P(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. Also, note that the underlying state transition distribution is not always explicitly known as it depends on the network dynamics, which is the model-free RL setting. Since both the individual reward R_i and the collective network reward R depend on the state transition distribution $\boldsymbol{\mathcal{T}}$, and user i's activity can be moderated both by actions taken with respect to i and actions taken for her peers, it is important to consider the effect of network structure ($\mathbf{G}$) when learning the policy π.

In a fully observable environment, all agents have access to the complete network state $\mathbf{s}_t$, and thus, we can perform *centralized learning and centralized execution*. In centralized learning, the joint state of all agents $\mathbf{s}_t$ is considered at the time of training. In centralized execution, the actions for an individual agent are based on the joint state of all agents, i.e., $a_{t,i} = \pi(\mathbf{s}_t)$. Specifically, the goal is to learn a policy that maps the network state at time t to the collective actions for each user, i.e., $\pi : \mathbf{s}_t \rightarrow \mathbf{a}_t$ such that the total expected discounted reward $\mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} R_t]$ is maximized, where $\gamma \in [0, 1)$ is the discount factor governing the relative importance of the immediate reward compared to future rewards.

In partially observable environments, agents do not have access to the complete state of the network, and each agent receives a partial observation of the environment $\mathbf{o}_i$ that depends on her local network structure or ego-network $\mathbf{G}_i$. Each user has a different ego-network, and thus, a user i receives a different observation than that of another user j in the network, i.e., $\mathbf{o}_i \neq \mathbf{o}_j$. A user i receives a local reward R_i, that can either depend on all peers in her local network $\mathbf{G}_i$, or a subset of peers in $\mathbf{G}_i$. Since, each user receives a different observation and reward, centralized learning and

centralized execution cannot be employed that involves sharing the state and actions between users, and a common reward. Thus, we consider *decentralized training and decentralized execution* in such environments. In decentralized training, each agent i learns her local policy π_i individually without sharing any state or observations with other agents. Decentralized execution means that the actions for an individual user are based solely on her individual state, i.e., $\mathbf{a}_{t,i} = \pi(\mathbf{s}_{t,i})$. The goal for each agent i is to learn her own policy π_i that maps her state at time t to her actions, i.e., $\pi_i : \mathbf{s}_{t,i} \rightarrow \mathbf{a}_{t,i}$ such that her total expected discounted local reward $\mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} R_{t,i}]$ is maximized.

Typically social networks are sparse with only $O(N)$ interactions between agents. Thus, the challenge is to capture $\sim N^2$ agent dependencies given these sparse interactions.

3.3 Applications

There are multiple applications that involve encouraging pro-social behavior in people. We can use Social RL to develop efficient solutions for these settings. Below we provide examples of these applications, and map those to the Social RL setting with respect to the social network relations, interactions, state, action and reward.

3.3.1 Fake News Mitigation

- Network: Nodes are users of a social network, and there are edges between two users if they have friends or followers relation
- Interaction: Users interact via activities such as making tweets, retweets and providing likes, for different types of news (true/fake)
- Agent's State: The number of posts (tweets and retweets) made for true and fake news, and likes received by a user

- Agent's Action: Increase the sharing rate for tweets and retweets of a user, for true news
- System Reward: Correlation between the number of exposures to fake and true news across all users
- Individual (Local) Reward: Correlation between the number of exposures to fake and true news in the local neighborhood

3.3.2 Viral Marketing

- Network: Nodes are users of a social network, and there are edges between two users if they have friends or followers relation
- Interaction: Users interact via activities such as posts and likes
- Agent's State: The number of posts made for a particular brand, and the number of likes received
- Agent's Action: Promote (or make more visible by increased sharing) the posts related to the specific brand
- System Reward: Brand awareness measured as the number of people reached, i.e., exposed to the posts related to the brand
- Individual (Local) Reward: Visibility measured as
  - Number of people exposed to a user's posts
  - Position (rank) of user's posts in her peers' feeds
  - Amount of time for which the user's feed stays at top in her peers' feeds

3.3.3 Recommendation Services with community interactions

Retail services (e.g., Amazon, eBay) can enhance their recommendations by utilizing the user interactions via reviews provided by users on purchased products, in order to increase the overall number of returning visitors, and customer satisfaction. The idea is to use the ratings/comments as feedback from users who have purchased similar products in the past to enhance the quality of one's reviews in the future. A user can utilize the comments/ratings provided by her neighbors (i.e., users who have also purchased similar products in the past) on her reviews, as feedback, to learn to make better reviews in the future. The system can utilize peer feedback to learn *efficient* users, i.e., who provide quality reviews that help other users, and increase the number of purchases or returning visitors, or customer satisfaction.

- Network: Nodes are users of the service, and there are edges between two users if they purchased similar products in the past
- Interaction: Users interact via providing comments, ratings, upvotes, downvotes on each other's reviews
- Agent's State: The number of reviews a user has provided on purchased products, and the number of upvotes, downvotes received
- Agent's Action: Increase the quality of reviews, e.g., adding more information
- System Reward: The system goal is to encourage users to give more qualitative reviews in order to improve
  - Number of purchases made, i.e., revenue
  - Customer satisfaction measured as number of returning visitors
- Individual (Local) Reward: Number of people who purchased the products reviewed by the user, after reading/reacting to her reviews

3.3.4 Online Discussion Forums/Question-Answer Services

- Network: The nodes are users of the service, and there are edges between two users if they take the same course (e.g., on Piazza, Blackboard), or discuss similar topics (e.g., on Quora, Stack Overflow)
- Interaction: Users interact via providing comments/ratings on each other's answers
- Agent's State: The number of questions a user has answered, and the number of upvotes/likes, downvotes received
- Agent's Action: Increase the quality and rate of providing answers to questions asked by peers
- System Reward:
  - Rate at which users receives answers, or the response time, on an average
  - Total number of unanswered questions (to be minimized)
  - Number of customers for the service (or class participation)
- Individual (Local) Reward:
  - Credits (e.g., reward points or 'badge' or class participation grade) received for providing answers.

3.3.5 Healthcare Applications

Some recent work has studied that online peer networks can motivate people to exercise more (e.g., [49]). They observed that social influence from online peers was more successful than promotional messages for increasing physical activities. Based on this, our idea is that if a user is performing well (achieves more milestones), then sharing her activities with those users who have similar health condition or physical

state can encourage the latter to also achieve more milestones, resulting in an increase in the reward defined as the total number of people with reduced obesity levels.

- Network: The nodes correspond to users of the application, with an edge between two users if they have same gender and similar age and body weight. These attributes, i.e., gender, age, body weight, are known only to the user and the system, and not shared with other users due to privacy concerns
- Interaction: Users interact via sharing their activities and providing comments or likes
- Agent's State: Number of miles walked, number of calories consumed
- Agent's Action: Increase the sharing rate of her activities e.g., milestones completed, miles walked, calorie intake, or some information/suggestions on weight loss, with her peers
- Network Reward:
  - Total number of people with reduced obesity levels
  - Average time taken for the people to lose weight
- Individual (Local) Reward: Number of people with reduced weight after reacting to user's activities

3.4 Challenges

We present the challenges that make Social RL problems difficult and potential ways to address those.

3.4.1 High Dimensionality

Much of the work on Multi-Agent Reinforcement Learning (MARL) is limited to small number of agents (< 50) (e.g., [52]). The standard approaches to train an

independent (decentralized) model for each user, i.e. N different model (e.g., [35]) are impractical for thousands of agents, especially when the policy function approximators are complex such as Deep Neural Networks. Moreover, joint (centralized) learning considers the actions of all agents for learning the policy of any agent, i.e., $\sim N^2$ dependencies. However, this is computationally intensive as the joint action space grows larger with an increase in the number of agents. This is particularly more challenging for Social RL due to curse of dimensionality, i.e., we have only $O(N)$ agent interactions available to model $\sim N^2$ dependencies. Thus, high dimensional spaces result in high computational cost and large variances in policy estimates.

We can utilize the social networks structure, characteristics of user interactions, and correlation between agents to develop policy learning approaches based on certain approximations, that simplify the learning process, while still accounting for all agent dependencies given large number of users. For example, [43] decoupled the different processes governing user interactions to approximate the joint action space more efficiently. They dynamically optimize the policy corresponding to agent interactions via tweets, but use historical data to estimate another type of interaction, i.e., feedback. This helps to reduce the number of parameters and avoid noisy policy estimates.

3.4.2 Sparsity

Typically, social networks are sparse with only $O(N)$ agent interactions. Thus, learning accurate policies that can capture $\sim N^2$ dependencies between agents, becomes challenging since there are not sufficient samples to learn from that can capture higher-order relations (interactions) in the data. To overcome this, we can utilize the symmetry between states and/or agents to reduce the size of the MDP as in [53, 54]. Furthermore, we can offset data sparsity issues by aggregating the interactions of similar/correlated users. This can also help to reduce the dimensionality of the search space and thus, the number of parameters and variance.

The sparsity of agent interactions also results in noisy policy estimates. To overcome this, [43] modeled user feedback that measures the efficacy of different users, and uses it as a signal to learn effective incentive allocation strategy. This can be thought of as a *reward shaping* technique which is used in multi-agent credit assignment and resource allocation problems where it is important to determine the contribution of each agent towards the common system goal for learning better policies [55]. However, their approach is different from standard reward shaping techniques, that consider a separate reward for each user (e.g., [35]). Since that requires a separate model for learning each agent's policy function, it is computationally intensive for large number of agents. To avoid this issue, they provide user feedback as input to the policy function approximator.

3.4.3 Partial Observability

The above problems become even more challenging in partially observable environments, where agents do not have access to the complete state of the environment and receive only a partial observation of the environment. Agents need to act solely based on their local observations, and can also receive different local rewards based on their individual actions. Thus, each agent needs to learn her own policy function that determines the actions that she needs to take in a particular state, to maximize her individual (local) reward. Markov Decision Processes (MDP) cannot be applied directly to partially observable domains, and instead Partially Observable MDPs (POMDP) are employed, that generally involve learning decentralized policies for agents (e.g., [56]) or making some approximations (e.g., [57]). However, the former is computationally infeasible for large number of agents, and for the latter, we need to ensure that the assumptions are consistent/preserve the network structure, and agent dependencies.

The challenge is that individual policies need to account for dependencies throughout the network, as the actions of agents are influenced by the activities of their peers

as well. Centralized learning and execution that helps to capture network-wide inter agent dependencies cannot be employed when agents receive different local rewards and observations. Moreover, decentralized learning is also infeasible for large number of agents, since it is impractical to learn thousands of complex policy functions. Also, in Social RL problems, the number of samples per user for learning individual policies are insufficient, due to the sparse interaction data, resulting in large errors due to variance. Thus, there is not sufficient local information available to capture network-wide dependencies.

The Partially Observable Environment can be further categorized as *Strongly Partially Observable Environment* and *Weakly Partially Observable Environment*, described as follows.

Weakly Partially Observable Environment

In most of the MARL approaches for partially observable domains that consider both local state and local reward, the hidden state of the environment becomes available to agents within a short period of time, i.e., before the time-horizon of the task. We refer to these environments as Weakly Partially Observable Environments. For such environments, the relevant state information can be used as history for policy learning (e.g., [58–61]), since the state information becomes available before the finite time-horizon. Additionally, these approaches consider a small number of agents, and hence the memory requirements for storing the history of all agents is lower.

Strongly Partially Observable Environment

There are scenarios where the state information does not become available to agents before the finite time-horizon of the task, and we refer to this as Strongly Partially Observable Environment. For e.g., in social networks, due to the directed nature of user interactions, a user cannot observe the state of her followers (e.g., in [44]). Moreover, social networks are not strongly connected, so even with message

passing the complete state of all followers would likely not be available before the finite time-horizon. Even if the state information could become available to agents, it is space-prohibitive to store the complete trajectory information for a large number of users. Thus, the relevant state information cannot be utilized by the user as history, which makes policy learning even more challenging in these environments. To address this problem, we can utilize the social network structure and user relations, to estimate the hidden environment state from the observed state of the environment and use it to improvise policy learning.

3.4.4 Evaluation

Another important challenge is to improvise the scheme of evaluating the learned policies. On-policy evaluation is relatively easier for MARL tasks with small number of agents, especially games. However, it might not be feasible always to apply the policy online before testing in certain real-world complex tasks, especially in medical and healthcare domains. This is also challenging for problems that involve online social networks because of multiple factors such as restrictions imposed by the operators of social networking websites, dynamic behavior/response of large number of users, government regulations. Additionally, in cases when it is feasible to conduct on-policy evaluation (e.g., in [15]), it is important to ensure that the subset of agents considered is representative of the entire population and is not biased. E.g., making real-time interventions on Twitter is not feasible, and thus [43] did not explicitly test if there is a reduction in fake news spread due to the learned policy. Instead, they used a simulated environment as a proxy for online interventions to measure the reward. In addition to simply using the simulated synthetic data, in scenarios where we cannot conduct online experiments, we need to find efficient ways that assess different methods using some held-out real-world data. For example, apart from computing the reward on simulated data, [43] also measured the impact/efficacy of agents selected on a held-out real dataset to evaluate their resource allocation strategy.

Moreover, the existing metrics to evaluate MARL approaches are based primarily on the reward. However, we believe that evaluation based solely on reward serves merely as a proof of concept as the model is designed to maximize the reward itself. Thus, it is also important to perform evaluations in environments where the assumptions made by the model do not necessarily hold. [43] conducted experiments by applying the learned policy with respect to different fake news diffusion processes based on different network properties such as degree distribution, clustering coefficient, centrality. They tested across multiple scenarios that are likely to hold in real world to assess generalisation to other environments.

In this work, we present potential solutions to address the above challenges using properties of the social network structure, agent interactions and correlations.

4 RELATED WORK

4.1 Relational Reinforcement Learning

Relational RL combines RL with relational learning or inductive logic programming ([62]) to represent states, actions, and policies using the structures and relations that identify them [63]. The structural representations allow to solve problems at an abstract level, and thus Relational RL approaches provide better generalization. Relational RL has been used in multi-agent systems to share information among agents, to learn better individual policies [64]. Previous work used first-order representations to achieve effective state/action abstractions (e.g., [63, 65, 66]). Some recent work has also used relational interactions between agents for policy learning in multi-agent systems (e.g., [67–70]). However, relational learning adds extra complexity with increased state space, due to the informedness/abstraction and generalization abilities provided by it [64]. Thus, these approaches are limited to discrete spaces with small number of agents that have dense interactions between them. We take motivation from relational learning to capture agent relations and interactions, however, we do not directly use Relational RL due to the above limitations.

4.2 Multi-Agent Reinforcement Learning

The key factors that differentiate Social RL from traditional MARL are the large number of agents and the sparse interactions between them, i.e., the network $\mathbf{G}$ that characterizes agent interactions. In addition, the interactions are dynamic, and thus the network can evolve with time. Agents can also interact via multiple activities, and each activity leads to the formation of a new link between agents. Thus, there exists d *links types* between agents corresponding to d different activities over the

Table 4.1.: Comparison of Social RL approaches (highlighted) with existing MARL approaches. '$-$' represents that a given attribute is not defined for the problem setting considered in the corresponding approach. The values in columns Maximum Dimensionality and Effective Dimensionality refer to a rough upper bound (i.e., $\sim$). Note: $\mathcal{K} \ll N$.

Approach	**Data/Environment**					**Model**				
	Network Density	**# Link Types** (d)	**Space**	**Observability**	**Reward Type**	**Maximum Dimensionality**		**Effective Dimensionality**		**Learning Method**
						# Policies	**# Actions Per Policy**	**# Policies**	**# Actions Per Policy**	
[15]	Sparse	1	Continuous	Full	Global	1	N	1	N	Centralized
[44]	Sparse	2	Continuous	Partial (Strong)	Local	N	1	N	1	Decentralized
Part 1	Sparse	2	Continuous	Full	Global	1	N	1	N	Centralized
Part 2	Sparse	3	Continuous	Full	Global	1	N^2	1	$\mathcal{K}^2$	Centralized
Part 3	Sparse	3	Continuous	Partial (Strong)	Local	N	N	1	N^2	Partially Centralized
[71]	Dense	1	Discrete	Full	Global	1	N	1	N	Centralized
[57]	Dense	1	Discrete	Full	Local	1	N	1	N	Centralized
[72] [73]	–	1	Discrete	Partial (Weak)	Local	N	N	N	N	Centralized
[74]	–	1	Discrete	Full	Global	1	N	1	$\mathcal{K}$	Centralized
[75]	–	1	Discrete	Partial (Weak)	Local	N	N	$\mathcal{K}$	$\mathcal{K}$	Decentralized

network. For e.g., when a user tweets a news, it reaches her followers leading to an interaction between users, and when a user provides feedback (e.g., by *liking* a tweet), that corresponds to another type of interaction ([43, 44]).

We present a comparison of Social RL with the state-of-the-art MARL approaches, based on different environment factors and model settings in Table 4.1. Social RL problems have a sparse network density (i.e., $O(N)$ agent interactions), whereas other MARL problems consider a dense network, that makes it easier to capture $\sim N^2$ agent dependencies. We consider the type of action search space, that can be either Discrete or Continuous. Search space is an important factor as it affects the computational cost. Learning policies for discrete spaces is easier than that for continuous spaces, and much of the RL models can only be applied to discrete spaces (e.g., [52, 57, 58, 61, 76–95]). The state-of-the-art Deep Q-Network (DQN) [96] is applicable only to discrete and low-dimensional action spaces, as it considers a finite set of actions, and finds the one that can provide maximum expected return in a given state. It can be extended to high-dimensional continuous state spaces via discretization, however, this can lead to multiple problems, especially, the curse of dimensionality [97]. Some recent work has considered continuous actions spaces, e.g., [56, 68, 98–105]. However, these methods are designed for small number of agents, and do not scale for thousands of agents. Moreover, many methods learn an independent model for each agent [56, 76, 105–115]. This is done to so that the size of search space is smaller (as opposed to the large search space in centralized learning), and the model can be trained efficiently. Social RL problems generally have continuous spaces ([15, 43, 44]) describing the complex user interactions and network activities in social networks, and the solutions for these can be easily extended/applied to the discrete problem settings.

The computational cost depends on the number of policies required to learn the actions for all agents. Based on the environment characteristics and the agent dependencies considered, each model has a maximum number of policies required to learn the actions for all agents, that we refer to as the Maximum Dimensionality in the table. And, we use *Effective Dimensionality* to refer to the effective (actual) number

of policies and actions learned based on the approximations/assumptions made by the model. For example, [74] reduces the effective number of actions to a constant $\mathcal{K}(\ll N)$ based on some aggregate of agents' actions. However, their approaches is designed for small restricted discrete action spaces. [75] approximated the effective number of agents as a constant $\mathcal{K}$ and learned $\mathcal{K}$ decentralized policies.

Agents can have either complete system information, i.e., Full Observability or partial information about the system, i.e., Partial Observability. In the latter case, each agent has a local view of the environment and makes decisions based only on her local observation. Some previous work considered fully observable setting (e.g., [15, 43, 44, 71, 77, 78, 98–101, 116, 117]). In general, partial observability is more challenging due to the lack of information available to learn accurate policies, compared to full observability. In addition, much of the previous work that considers partially observable environments, has considers a weakly partially observable environment (described in Sec. 3.4.3) (e.g., [72,75]), where the hidden state of the environment can be used as history to improvise policy estimates. Some recent approaches have developed solutions for strongly partially observable environment, where the latent state of the environment is not available to agents before the time-horizon of the task (e.g., [44]), and thus, cannot be used as history for policy learning. However, these solutions do not scale for large number of agents.

Furthermore, the reward received by an agent in the system can be either Global, i.e., common reward for all agents based on their collective state and actions, or it can be Local, i.e., reward for the individual agent based solely on her individual actions and/or actions of a subset of other agents in the system. For example, [44] considers a Social RL setting where a user receive local reward based on the rank of her posts in her followers' news feed. Cooperative MARL tasks have a common reward for all users (e.g., [89,102,118]). Decentralized (or independent) Learning considers learning an independent policy function for each agent based solely on her individual state and/or local reward, without conditioning on the collective state of all agents. Thus, the dimensionality for decentralized learning is $\sim N$. Some decentralized learning

approaches (e.g., [89,118]) assume a global state shared across users and learns actions conditioned on the global state.

Much of the work in MARL has considered decentralized learning, e.g., [56–58,90–92,105,106,108,110,111,114,115,119–127]. [107] proposed an Independent Q-Learning (IQL) approach that learns a Q-function for all state-action pairs for all agents in the system. However, this is applicable only for small number of agents, and results in policies with large variance [97], especially when there are sparse/insufficient samples. [128] considered the problem of energy sharing optimization where each building (agent) learns to choose from a discrete set of actions. A DQN is learnt for each building. However the number of agents are limited to only ten, and the model does not scale for larger number of agents [97]. [93] learnt an independent DQN for each agent in the game of soccer with discrete low dimensional state and action spaces, for only five agents. [113] considered a common-pool resource problem where each agent learns a self-interested policy via DQN. [129] introduced the problem where agents while learning the main policy, also learn to trade in their respective actions in exchange for the environmental reward. However, they consider only two agents. [130] executes an asynchronous advantage actor-critic algorithm to learn policy for each agent in parallel, however the number of agents again is limited to four. [60] used Long Short Term Memory (LSTM) to develop Deep Recurrent Q-Network (DRQN) that extends Deep Q-Networks and uses history to overcome the problem of hidden information in partially observable environments. [61] extended DRQN to solve POMDPs for multi-agent systems and proposed Distributed Deep Recurrent Q-Network (DDRQN) which learns an independent DRQN for each agent, and is applicable only to a small number of agents with discrete state/action spaces. [58] extends MARL to multi-task setting, where partial observability is realized by hiding task identity from agents, who learn to cooperate to solve decentralized POMDP tasks. They learn a decentralized policy for each agent, however, their approach cannot be extended when agents receive different rewards [97]. To address the problem of scaling to large number of agents in partially observable domains, [102] proposed to first learn for sub-tasks that involve

small number of agents, and then add more agents for more complex tasks. However, this approach can be applied only to problems that can be divided into sub-problems with smaller number of agents.

Centralized (or joint) Learning considers the joint states/observations of all agents, i.e. $\sim N^2$ dependencies to learn a joint action over all agents. Much of the previous work considered centralized learning with small number of agents (e.g., [77–79, 98–100, 116, 117]), and some recent approaches have considered centralized learning with large number of agents (e.g., [15, 71]). However, the computational cost increases with the number of agents since we need to learn $\sim N^2$ parameters to model the agent dependencies, and centralized learning can only be used when the environment is fully observable to all agents.

Some approaches have used centralized training and decentralized execution (e.g., [72]) for partially observable domains. The idea is to learn a centralized controller based on the actions and observations of all users that guides the policy learning of decentralized agents who can then execute individual actions independently based solely on their local observations. [131, 132] developed a master-slave multi-agent RL solution in which a a master agent receives and collectively processes messages from slave agents. It then provides unique instructive messages to each slave agent. Slave agents use their own information and the instructive messages from the master agent to decide which action to take. [85, 133] proposed solutions for multi-agent credit assignment problems in a partially observable setting where it is difficult for agents to realize their contribution to the team's success from global rewards. They propose to estimate a baseline function using a centralized controller, that is used to obtain the expected reward for each decentralized agent. However, their model can be applied only for discrete action spaces and small number of agents [97]. [134] proposed deep loosely coupled Q-network for partially observable systems where each agent has a degree of independence and she can choose to either learn independently or cooperate jointly, based on her local observation. However, these approaches are limited to small number of agents. There are also scenarios when agents receive noisy

observations and thus, each agent has a different observation of the true state [97]. [135] introduced learning of a communication policy that allows agents to interact and share their observations. Along with learning the main policy, agents need to determine whether their observations are informative to share with other agents. However, learning the communication protocols along with policy learning increases the number of parameters, and adds extra time complexity for execution.

4.3 Process Interventions

Previous work considered modeling news diffusion using Independent Cascade (IC) model [12–14] and tried to limit the influence of fake news by selecting subset of users that can spread true news. However, they do not differentiate between (re)tweet and exposure events, and consider a user to be *activated* in both cases. Moreover, their assumptions for user interactions and information diffusion, under IC model, do not hold in real-world social networks [136], and without these assumptions, their approach is computationally infeasible as shown by them. They consider a user can be activated only once, and users can be activated only by a single user. But, in practice, a user can become active or inactive multiple times, depending on their interactions with other users, that are mutually-exciting in nature [15]. Additionally, they consider fixed (re)tweeting rates, however, this does not hold in practice [136]. Also, they assume that a user can be exposed to either fake or true news campaigns (not both), and once activated, users cannot change campaigns.

Multivariate Hawkes Process (MHP), considers history of user events and interactions, and better capture news diffusion, They consider self and mutually exciting nature of user activities in real-world networks [43]. Thus, instead of assuming fixed probabilities for news sharing (e.g., in [12]), we model excitation events using MHPs and integrate those in a RL framework to capture the dynamically changing user behavior. Moreover, we estimate base intensities (natural rate) of users to spread fake and true news, from real data, instead of assuming any pre-determined rates. We do

not make any strong assumptions as in IC model (e.g., [12]), and in our case users can be activated by both fake and true news campaigns, as in a real-world setting.

5 CENTRALIZED SOCIAL REINFORCEMENT LEARNING

5.1 Introduction

In this work, we consider the task of combating fake news dissemination in online social media systems. Under the assumption we can characterize the diffusion of news over the network by some stochastic process, and that the diffusion of true news is independent from the diffusion of fake news, our aim is to mitigate the spread of fake news by increasing the spread of true news.

We have a fixed budget that can be provided as incentives and thus, appropriate selection of users and efficient allocation among those is important. The response a user receives on sharing some post is an important indicator of her effectiveness in spreading the news further, and can help to determine the amount of incentive to spend on the user. For example, in social networks, this response can be quantitatively measured in terms of number of "likes" received by the user.

Social reinforcement refers to the process where acceptance and praise from others reinforces behaviors/preferences of an individual (see e.g., [18]). We propose to model feedback from peers to learn better incentivization policies. Rewards on social media (i.e., 'likes') are a form of acceptance and appreciation from peers, which affects the regions of the brain responsible for decision-making and thus leads to a change in their behavior [137, 138]. Specifically, we use the number of 'likes' obtained on sharing a post users provide a positive reinforcement by hitting the 'like' button as observed in [19]. 'Likes' have also been used as an important feature in classification of news as fake or true [139].

To learn how to efficiently allocate incentives, we consider estimates of user *feedback* and user *political bias*. Since the response a user provides for a tweet is likely to depend on her degree of *political bias*, we conjecture that estimates of user response

(as a function of political bias) can help to efficiently select people to incentivize to promote true news. To incorporate these effects, we consider a user's leaning towards the Democratic and Republican Parties. Thus, we model the feedback as a function of political bias, towards the Democratic and Republican Parties. Specifically, we model user response using a Multivariate Hawkes Process (MHP), whose base intensity is proportional to their political bias, and interleave it with the news diffusion processes (also modeled as an MHP). We estimate a user's initial political bias using a community detection algorithm and propose a model to update the bias over time.

Our setting is a cooperative multi-agent RL problem (MARL), where the number of agents is large and the state and action spaces are continuous, which makes the problem more challenging. Much of the previous work in MARL focuses on learning a separate model for each user independently, or learning jointly by considering the full state and action spaces across all users. However, both these approaches are computationally intensive for a large number of agents. We avoid this by decoupling of the post and response processes to approximate the joint action space more efficiently. We dynamically optimize the intensity for the MHP corresponding to post events, but only estimate parameters for the response events from historical data. By doing this, we reduce the number of parameters and avoid noisy policy estimates.

To evaluate the performance of our model, we use two real-world Twitter datasets. Since we have access to limited real-world data, and we cannot make real-time intervention, we perform experiments on semi-synthetic data demonstrating the results with respect to different network properties for fake and true news diffusion likely to hold in real-world. The results show that adding intervention to increase the spread of true news is beneficial for mitigating the impact of fake news relative to providing no incentive. And compared to other baselines that do not consider estimates of user response and political bias, our model is able to achieve increased true news diffusion, in terms of maximizing the number of people reached and the number of *mitigated* users, that is users who are already exposed to fake news, that become exposed to true news.

5.2 Related Work

[15] proposed to mitigate the impact of fake news by making interventions to true news diffusion process modeled as MHP, and mapping the problem to a Markov Decision Process (MDP). Our work is motivated by their approach, but we extend their model to incorporate a feedback component between pairs of users modeled using a separate MHP, and interleave it with the news diffusion MHP. We believe that feedback is important in selection of users for efficient incentive allocation under budget constraints. The feedback provided to users can be thought of as a *reward shaping* technique, which is used in multi-agent credit assignment and resource allocation problems where it is important to determine the contribution of each agent towards the common system goal for learning better policies [55]. However, our approach is different from standard reward shaping techniques, which consider a separate feedback for each user in the reward function. Since that requires a separate model for learning each agent's policy function, it is computationally intensive for large number of agents. To avoid this issue, we provide user feedback as input to the policy function approximator.

[44] uses deep reinforcement learning with marked temporal point processes for incentivizing agents in personalized teaching and viral marketing domains. Similar to our approach, they use feedback events to improve policy learning. However, their events are application specific and are assumed to be generated from a black box distribution. In contrast, we propose a process governing generation of feedback events, and evaluate it using events from real data. Moreover, [44] trains a separate model for each user *independently*, which is computationally intensive. In contrast, we *decouple* the news diffusion and response processes to learn an *approximate* model. This reduces the size of the joint action space and helps to avoid noisy estimates, in addition to reducing the number of parameters (compared to the full joint).

5.3 Problem Definition

We consider the following setting. Let there be N users and let $\mathbf{G}$ represent the followers network, where $G_{ji} = 1$ if i follows j, and 0 otherwise. We consider tweets corresponding to news stories, labeled fake (F) or true (T). We consider the act of tweeting and retweeting by users as a news sharing event and do not differentiate between them. The data contains a temporal stream of events $e = (t, i, h)$, where t is the time-stamp at which user i (re)tweets a post with label $h = F$ or T corresponding to fake or true news. Let $\mathcal{N}_i(t, h)$ be the number of times user i shares posts corresponding to $h = F$ (fake) or T (true) news, respectively up to time t. Let $\mathcal{T}$ be the time-horizon of the task, divided into K stages of length Δ, where stage k corresponds to the time interval $[\tau_k, \tau_{k+1})$ such that $\tau_{k+1} - \tau_k = \Delta$.

The impact of fake and true news can be measured in terms of the number of people who are exposed, that has also been used in [7, 15]. We can compute the number of times a user i is exposed to news by time t as $\mathbf{G}_{.i} \cdot \mathcal{N}(t, h)$. Since it is difficult to stop the spread of fake news, we want to ensure that users receive at least as much true news as they do fake news (i.e., $\mathbf{G}_{.i} \cdot \mathcal{N}(t, F) \simeq \mathbf{G}_{.i} \cdot \mathcal{N}(t, T)$). We believe that an increased exposure to true news can increase skepticism for fake news, as described in Section 1. Note that our goal is not to detect, but to mitigate the impact of fake news. Thus, we consider the (re)tweets labeled fake/true apriori.

The objective is then to incentivize users to share true news in a targeted fashion such that the people who are exposed more to fake news are also exposed more to true news. From an algorithmic perspective, we want to learn how to efficiently allocate the incentives assuming a budget constraint. Specifically, given the *state* of the system $\mathbf{s} \in \mathbb{R}^{dN} (\mathbf{s} \geq 0, d \in \mathbb{Z}^+)$, which represents d network activities over N users, we want to learn an incentivization policy function $\pi : \mathbf{s} \to \mathbf{a}$ to obtain incentive *actions* $\mathbf{a} \in \mathbb{R}^N (\mathbf{a} \geq 0)$ corresponding to the increase in the likelihood of sharing true news per user. Since we will only incentivize sharing of true news, we evaluate our intervention strategy by computing the correlation between exposures

to fake and true news. We also measure the distinct number of people mitigated and assess the effectiveness of users selected by the strategy to spread true news.

5.4 News Diffusion Processes

A number of diffusion models have been developed to capture the spread of information in social networks. Many of these models are based on stochastic processes that use *intensity functions* governing the sharing rate per user. Some intensity functions depend only on network structure, while others take into account the effect of previous events and interactions between users. We considered several alternative processes and evaluate which better characterizes the diffusion of news in our real data.

Generative Process Since the process of fake and true news diffusion is the same except for parameters, we provide a generic expression for intensity. Let $\lambda_{h,i}$ be the intensity for user i sharing a post, where $h = T$ corresponds to true news, and $h = F$ corresponds to fake news. The generative process to determine time-stamps at which user i makes posts corresponding to true news, given their respective intensities, is described as follows. Let $\{t_{h,i,j}\}_{j\geq 1}$ be the time-stamps for user i. Define $\{\sqcup_{h,i,j}\}_{j\geq 1}$ to be the inter-arrival times, which are assumed to be independent for all processes. Assuming the diffusion processes start at time 0, we can write, $t_{h,i,m} = \sum_{n=1}^{m} \sqcup_{h,i,n}$. $\mathcal{N}_i(t,h)$ is the number of times user i shares posts by time t (defined in Section 5.3). We have $\mathcal{N}_i(t,h) = \sum_{m\geq 1} \mathbb{I}(t \geq t_{h,i,m})$. Let $\kappa_{h,i}$ be the fraction of news tweets up to time $\mathcal{T}$ by user i for $h = T$ (true) or $h = F$ (fake) news. These values are computed from historical data. The sampling method to generate inter-arrival times depends on the type of diffusion process and is explained for each type below.

5.4.1 Diffusion based on Network Structure

DEG Intensity depends on the number of followers and followees of the user: $\lambda_{h,i} = \kappa_{h,i}(\sum_{u=1}^{N} G_{iu} + \sum_{u=1}^{N} G_{ui})$

CEN Intensity is proportional to closeness centrality [140]. Let δ_{iu} be the shortest distance from i to u in $\mathbf{G}$: $\lambda_{h,i} = \kappa_{h,i}(\sum_{u=1}^{N} \delta_{iu})^{-1}$

Generative Process The above processes are homogeneous poisson processes, whose inter-arrival times are exponentially distributed, $f_{h,\sqcup_i}(t) = \lambda_{h,i} e^{-\lambda_{h,i} t}$, with inverse cdf is given by $F^{-1}_{h,\sqcup_i}(u) = \frac{-\ln u}{\lambda_{h,i}}$. Since $F^{-1}_{h,\sqcup_i}(t)$ has a closed form expression, we use inverse transform sampling to sample $\sqcup_{h,i,j} = \frac{-\ln u_j}{\lambda_{h,i}}$, where $j \geq 1, u_j \sim \mathcal{U}(0,1)$. After we obtain the inter-arrival times $\sqcup_{i,j}$, we can generate the event times $t_{h,i,j} = \sum_{n=1}^{j} \sqcup_{h,i,n}$.

5.4.2 Diffusion based on History and Influence

We consider an N-dimensional MHP, where each dimension corresponds to a user i. MHP naturally captures the phenomenon of self and mutual excitations between user events discussed in Sec. 4.3.

$$\lambda_{h,i}(t) = \mu_{h,i} + \sum_{j=1}^{N} \int_0^t \Phi_{ji} \; (\omega_h e^{-\omega_h t}) \; d\mathcal{N}_j(s,h)$$

where, the integral is over time, and s is used as placeholder for limits $\{0, t\}$. $\mu_{h,i}$ is user i's base exogenous intensity. The second term considers the effect of previous events and mutual excitations among users, where $\mathbf{\Phi}$ is a kernel adjacency matrix and Φ_{ji} corresponds to the impact that user j has on user i in the news diffusion process. We use the standard Hawkes exponential kernel $\omega_h e^{-\omega_h t}$ to capture the decaying effect of history over time, where ω_h is the hyper-parameter governing the rate of decay. $\boldsymbol{\mu_h}$ and $\mathbf{\Phi}$ are estimated from real data.

Generative Process Ogata's Thinning Algorithm [141] is used to generate event times by sampling inter-arrival times using rejection sampling. The idea is to first generate events from a homogeneous poisson process with a rate greater than the desired rate, and then reject an appropriate fraction of events generated to achieve the desired rate [142]. After this, we assign a dimension $i \in [1, N]$ to each of the time-stamps generated with probability proportional to $\lambda_{h,i}$.

Efficient Computation of Intensity To incorporate the effect of past events, much of the recent work uses complete trajectories of users (eg. [44]). However, since we consider MHPs to characterize user activities in the network, our model can include the effect of self and other users' history and actions implicitly, as in [15] and [43]. The efficient computation of history and summing it to one scalar per dimension as described below, is computationally efficient as we do not need to store the complete trajectory for each user.

The diffusion of news is modeled as MHP, a non-Markovian process. However, since we map the problem to a Markov Decision Process (MDP), we need to include the effect of history from previous events. Let $\mathcal{H}_{F,i}$ be the effect of intensity due to all events in previous k stages, for user i, on the future $t > \tau_k$, for fake news diffusion. $\mathcal{H}_{F,i} = \omega_F \sum_{j=1}^{N} \int_0^{\tau_k} \Phi_{ji}\, e^{-\omega_F(t-s)}\, d\mathcal{N}_j(s, F)$. As observed in previous work ([143,144]), exponential kernel allows to efficiently compute the intensity, by defining $y_{F,k,i} = \lambda_{F,i}(\tau_k) - \mu_{F,i}$, so that, $\mathcal{H}_{F,i} = y_{F,k,i} e^{-\omega_F(t-\tau_k)}$. Hence, using $y_{F,k,i}$, we can efficiently compute the intensity at $t > \tau_k$, without having to sum over all previous k stages. Similarly, we have $y_{T,k,i} = \lambda_{T,i}(\tau_k) - a_{k-1,i} - \mu_{T,i}$ for true news diffusion.

5.4.3 Diffusion based on Political Bias

Apart from network properties and user interactions, we believe that user's political bias is an important factor governing the probability of her sharing a post. Hence we model political bias and its change over time based on [145]. The idea is that when two users interact, it changes their degree of bias. Let $\mathbf{G}_{i\cdot} = \{j | G_{ij} = 1\}$ be the set

Algorithm 1 Update Political Bias

1: **Input:** $\mathcal{I}_k, \{\mathbf{G}_{i.}\}_{i=1}^N, \{b_{k-1,i}\}_{i=1}^N$
2: /* Initialize bias values at stage k with those at stage $k-1$ */
3: **for** $i = 1...N$ **do**
4: $\quad b_{D,k,i} = b_{D,k-1,i}, \quad b_{R,k,i} = b_{R,k-1,i}$
5: **end for**
6: /* Update bias based on interactions during stage k*/
7: **for** $e = (t, i, h) \in \mathcal{I}_k$ **do**
8: $\quad$ Let i be the user corresponding to the event e.
9: $\quad$ **for** each $j \in \mathbf{G}_{i.}$ **do**
10: $\quad\quad$ **if** $b_{D,k-1,i} > b_{R,k-1,i}$ **then**
11: $\quad\quad\quad b_{D,k,j} = b_{D,k,j} + 0.5[|b_{D,k,i} - b_{D,k,j}| - \rho(|b_{D,k,i} - b_{D,k,j}|)]$
12: $\quad\quad\quad b_{R,k,j} = 1 - b_{D,k,j}$
13: $\quad\quad$ **else**
14: $\quad\quad\quad b_{R,k,j} = b_{R,k,j} + 0.5[|b_{R,k,i} - b_{R,k,j}| - \rho(|b_{R,k,i} - b_{R,k,j}|)]$
15: $\quad\quad\quad b_{D,k,j} = 1 - b_{R,k,j}$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end for**
19: **return** $\{b_{k,i}\}_{i=1}^N$

of followers of user i. Let $\mathcal{I}_k$ be the list of events $\{e = (t, i, h)\}_{\tau_k \le t \le \tau_{k+1}}$ that occurred during stage k, sorted in chronological order. Let $b_{D,k,i}$ and $b_{R,k,i}$ be the bias of user i, respectively, for stage k. We say that a user i, in stage k, has polarity $p_{k,i} = D$ if $b_{D,k,i} > b_{R,k,i}$, and $p_{k,i} = R$ otherwise. We assume that the bias is constant in the interval $[\tau_k, \tau_{k+1})$ and update it at the end of stage k (time τ_{k+1}), taking into account the cumulative effect of interactions during the interval, as described in Algorithm 1. The function ρ in lines 11 and 14 helps to maintain the bias values in [0, 1] as $\rho(x) = 0$ if $x \in [0, 0.5]$, and $\rho(x) = 1$ if $x \in (0.5, 1]$.

Aligned (AL) If a user i has polarity D in stage k, then her intensity for stage $k+1$ will be set to her bias (at stage k) for D, otherwise the intensity will be set to her bias for R: $\lambda_{h,k+1,i} = \mathbb{I}(p_{i,k} = R)\, b^R_{i,k} + \mathbb{I}(p_{i,k} = D)\, b^D_{i,k}$

BCM Similar to AL, but the bias at stage k is computed using Bounded Confidence Model (BCM) [41, 42] (Section 2.5) that has been widely used to capture opinion dynamics in social networks.

Generative Process Given the bias values computed for stage k, the diffusion process during stage $k+1$, for each user i, is a homogeneous Poisson process. Therefore, we sample the inter-arrival times as $\sqcup_{k+1,i,j} = \frac{-\ln \mathcal{U}(0,1)}{\lambda_{k+1,i}}$. For stage $k+1$, we can write $t_{k+1,i,j} = \tau_{k+1} + \sum_{n=1}^{j} \sqcup_{k+1,i,n}$.

5.4.4 Evaluation of Proposed Processes

Our goal is to quantitatively assess which of the above processes better characterizes news diffusion in real-world data. For this, we use a portion of the data as training data to infer parameters, and then simulate processes for later stages, with the assumption that parameters learnt from historical data (past stages) continue to describe the process in the future. We compare characteristics of the simulated data with the real data to evaluate the various processes.

We use two real-world datasets, Twitter 2016 and Twitter 2015 [5, 146], with 750 and 2050 users in the networks, respectively. We observed that in our data around 75% of the news last for 40 hours, and thus we take $\mathcal{T} = 40$ hours, with 40 stages of length $\Delta T = 1$ hour each. Also we observed that true news decays faster than fake news, and thus chose $\omega_F < \omega_T$. Specifically, we set $\omega_F = 0.6$ and $\omega_T = 1$ for Twitter 2016, and $\omega_F = 0.75$ and $\omega_T = 1$ for Twitter 2015. To decide ω, we performed a grid search and chose the one with least error. We consider ω as a pre-specified hyper-parameter as in [15], and did not estimate from data due to increased computational cost.

We infer parameters $(\boldsymbol{\mu}, \boldsymbol{\chi}, \boldsymbol{\Phi})$ using Maximum Likelihood Estimation as in [15, 147, 148] using data from first 10 stages. Suppose we have l previously observed sequences $\mathcal{I} = \{o_q\}_{q=1}^{l}$, where each o is a sequence of events $\{(t_j^o, i_j^o, h_j^o)\}_{j=1}^{n_o}$ observed during the first 10 stages, and n_o is the number of events in o. Since we assume that the diffusion of fake ($h_j^o = F$) and true ($h_j^o = T$) news is independent, we separate the events corresponding to fake and true news diffusion, and learn the respective parameters separately. We provide a generic expression for likelihood, with exponential kernel for MHP:

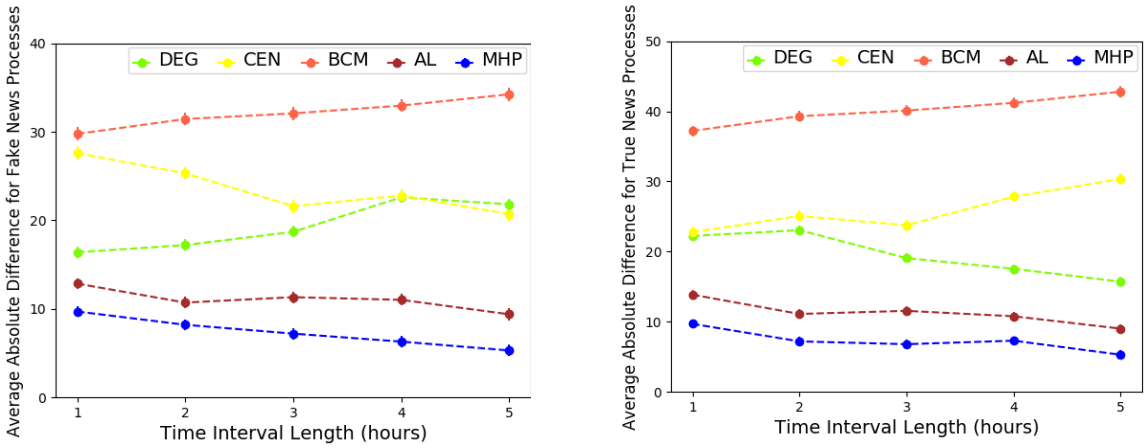
$$L(\boldsymbol{\Theta}) = \sum_{o \in \mathcal{I}} [\sum_{j=1}^{n_o} \log\ \lambda_{i_j^o}(t_j^o) - \sum_{i=1}^{N} \int_0^{T_{PE}} \lambda_i(t) dt)] \tag{5.1}$$

where T_{PE} corresponds to the time-stamp of the historical data used to estimate parameters (10 in our case). We minimize with L_1 regularization to avoid over-fitting,

$$\min_{\boldsymbol{\mu}, \boldsymbol{\Phi}} -L(\boldsymbol{\Phi}, \boldsymbol{\mu}) + \zeta_1 ||\boldsymbol{\mu}||_1 + \zeta_2 ||\boldsymbol{\Phi}||_1 \tag{5.2}$$

where $||\boldsymbol{\Phi}||_1 = \sum_{i,j=1}^{N} \Phi_{ij}$, is used to enforce sparsity of the matrix $\boldsymbol{\Phi}$. To efficiently solve our optimization problem, we divide it into easily solvable sub-problems, based on the approach of Alternating Direction Method of Multipliers (ADMM) [149]. See [147] for more details on efficient optimization.

We evaluate which of the proposed processes better captures the real data. The training/test framework is shown in Fig. 5.3. The simulations of MHP are performed using "tick" python library ([150]). Using the parameters learned from the first 10 stages, we simulate the process for later stages. Let $\mathcal{N}_{D,i}(t)$ be the number of events of user i up to time t in the real data, and let $\mathcal{N}_{P,i}(t)$ be the number of events of user i up to time t obtained from the simulating process P. $\mathcal{N}_{P,i}(t) = \mathcal{N}_i(t, F)$ for fake news diffusion and $\mathcal{N}_{P,i}(t) = \mathcal{N}_i(t, T)$ for true news diffusion defined in Sec. 5.3. For a given interval of length Δ, we define error $\mathcal{E}_{P,\Delta}$ as the absolute difference between the number of events generated from the simulated process P and the number of events in the real data in the interval Δ, averaged over all users:



(a) Fake News Processes

(b) True News Processes

Figure 5.1.: Difference (expected and observed number of events)

$$\mathcal{E}_{P,\Delta} = \frac{1}{N}\sum_{i=1}^{N} |[\mathcal{N}_{D,i}(t'+\Delta) - \mathcal{N}_{D,i}(t')] - [\mathcal{N}_{P,i}(t'+\Delta) - \mathcal{N}_{P,i}(t')]| \tag{5.3}$$

where $t' > T_{PE}(= 10)$. Figure 5.1 shows the error, for each process, corresponding to different values of Δ, where we average over 10 different time intervals for each value of Δ by taking different values of $t' \in [11, 40]$. We observe that MHP achieves the least error, and that it decreases with increasing interval length, for both fake and true news diffusion. This can be attributed to the fact that MHP considers history of previous events, and mutual excitations. Thus, we can say that MHP closely models the diffusion of fake and true news in real-world data, and use it as the process characterizing news diffusion in our model described next.

5.5 Incentivization Model

Let $\mathbf{s}_k$ be the state of the network at stage k. We define actions $\mathbf{a}_k \in \mathbb{R}^N$, where $a_{k,i} \geq 0$ is the incentive provided to user i to promote true news, during stage k. We learn the function $\pi : \mathbf{s}_k \to \mathbf{a}_k$ by using policy optimization problem in a Markov

Decision Process (MDP) ([151]), such that the reward (objective) defined in Sec. 5.5.2 is maximized. MDP based methods take into account the reward achieved on applying the policy, from the current stage as well as from the future stages. We add $\mathbf{a}_k$ as an intervention to the intensity function for true news diffusion modeled using MHP.

$$\lambda_{T,i}(t) = \mu_{T,i} + a_{k,i} + \sum_{j=1}^{N} \int_0^t \Phi_{ji} \ (\omega_T e^{-\omega_T t}) \ d\mathcal{N}_j(s, T) \tag{5.4}$$

where $\tau_k \leq t < \tau_{k+1}$ for the k^{th} stage. Since the total amount of incentive provided is usually limited, we impose budget constraint by fixing the sum of incentives for all users at stage k to be O_k, $(\sum_{i=1}^{N} a_{k,i} = O_k)$. We consider $\mathbf{a}_k$ as actions in the MDP, where the space of all possible actions is given by $\mathbf{A}_k = \{\mathbf{a}_k \in R^N | \mathbf{a}_k \geq 0, ||\mathbf{a}_k||_1 = O_k\}$.

Generative Process The process to generate events after applying intervention is the same as in Sec. 5.4.2, except the time-stamps are generated for every stage k using the corresponding intensity for the stage, similar to the diffusion based on political bias (Sec. 5.4.3).

5.5.1 State Features

We represent the state of the network $\mathbf{s}_k$ for stage k as $\mathbf{s}_k = (\mathbf{n}_k, \mathbf{w}_k)$. Here, $\mathbf{n}_k(T)$ and $\mathbf{n}_k(F)$ refer to the number of true and fake news events in the previous stage, respectively, and are obtained as $\mathbf{n}_k(h) = \boldsymbol{\mathcal{N}}(\tau_k, h) - \boldsymbol{\mathcal{N}}(\tau_{k-1}, h)$, where $\boldsymbol{\mathcal{N}}(t, h)$ is the number of events upto time t, corresponding to true ($h = T$) or fake ($h = F$) news diffusion, described in Sec. 5.3. $\mathbf{w}_k$ refers to user responses in terms of the number of likes received.

Number of Events in Previous Stages

As shown in previous work [15, 152, 153], a common choice of features to parameterize point processes is the number of events in the previous stage. Hence, we

define $\mathbf{n}_k(F) \in \mathbb{R}^N$ and $\mathbf{n}_k(T) \in \mathbb{R}^N$, such that $n_{k,i}(F) = \mathcal{N}_i(\tau_k, F) - \mathcal{N}_i(\tau_{k-1}, F)$ and $n_{k,i}(T) = \mathcal{N}_i(\tau_k, T) - \mathcal{N}_i(\tau_{k-1}, T)$.

User Response

We consider the news diffusion and response processes to be inter-leaving, and measure response for a user at the end of each stage. Let $\mathbf{W}(t) = [W_{ui}(t)]_{u,i=1,u\neq i}^{N}$, where $W_{ui}(t)$ is the number of times user i likes the (re)tweets by user u up to time t. $W_{k,u} = \sum_{i=1,u\neq i}^{N} \int_{\tau_k}^{\tau_{k+1}} dW_{ui}(s)$ is the total likes received by user u during stage k. Hence, the *feature vector* representing the feedback received by users is $\mathbf{w}_k = \{W_{k,u}\}_{u=1}^{N}$.

We cannot make real-time interventions on Twitter and do not know apriori the response (number of likes) a user would receive on (re)tweeting under the news diffusion model. Hence, we model the environment generating user responses using another MHP, motivated by [154] that modeled the number of times user i retweets source u. We extend their approach, in our case, to model the number of likes given by a user i to source u, by incorporating the stage (time) dependent political bias as the base exogenous intensity explained below. For each pair of users, we have corresponding intensity modeled using MHP, given as $\{\psi_{u,i}(t)\}, u, i \in [1, N], u \neq i)$.

Aligned Bias User Response

If source u and her follower i have the same political leaning (polarity), then the probability (intensity) of i "liking" u's post increases, whereas if they have different polarity, then the probability decreases. To realize this, we adjust the base intensity depending on the bias values.

$$\begin{aligned}\psi_{k+1,u,i}(t) = \chi_i + \mathbb{I}(p_{k,i}{=}p_{u,k}) b_{p_{k,i},k,i} - \mathbb{I}(p_{k,i}{\neq}p_{k,u}) b_{\neg p_{k,i},k,i} \\ + \sum_{j\in\mathbf{G}_{\cdot i}}^{N} \int_0^t \omega_{\mathcal{L}} G_{ji}\ e^{-\omega_{\mathcal{L}} t}\ dW_{uj}(s)\end{aligned} \tag{5.5}$$

where $t \in [\tau_k, \tau_{k+1})$. χ_i is the base intensity estimated from the data that is independent of the history. $\mathbf{G}_{\cdot i}$ is the set of followees of i. Using above, we try to accumulate the response a user receives from her direct and indirect followers, by aggregating the likes by followees of user i to the post of user u. The more frequently the followees of i like u's posts, the more she tends to "like" u's posts. When i likes u's post, $W_{ui}(t)$ gets incremented, furthur increasing the chances of liking u's post among the followers of i. We simulate the above process for each stage k by first computing the users u who shared true news during stage k, i.e., users with $n_{k,u}(T) > 0$, and then generate feedback events using $\psi_{k,u,i}$, only for those users. For simplicity, we set $\omega_{\mathcal{L}} = 1$.

To test whether bias helps to model the response process better, we compare it with the alternative model described below that doesn't consider political bias.

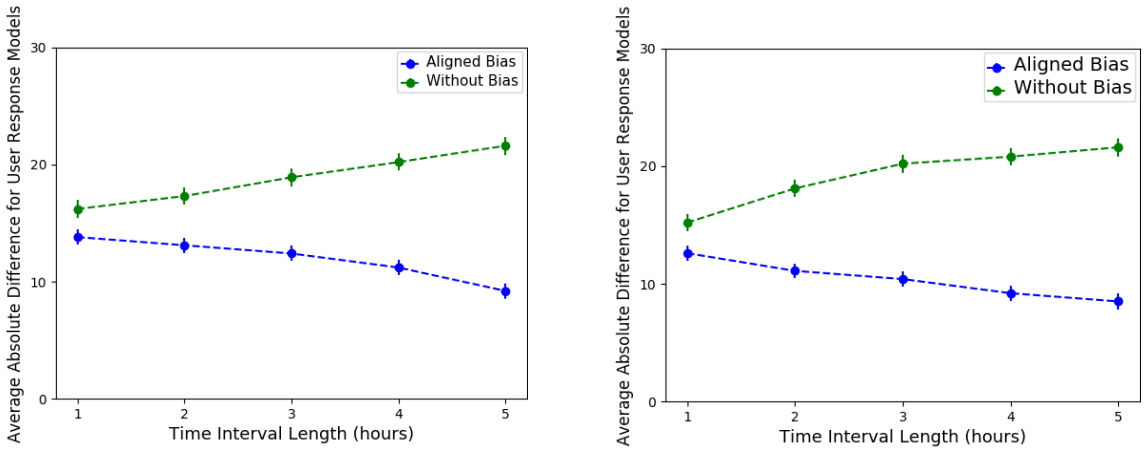
Without Bias User Response

$$\psi_{u,i}(t) = \chi_i + \sum_{j \in \mathbf{G}_{\cdot i}}^{N} \int_0^t \omega_{\mathcal{L}} G_{ji}\ e^{-\omega_{\mathcal{L}} t}\ dW_{uj}(s) \tag{5.6}$$

To evaluate how well the above model captures "like" events in the network, we use a similar setting as in Sec. 5.4.4, and observed that the Aligned Bias User Response model outperforms Without Bias User Response model that does not take into account bias. We compare the number of likes generated from the above models (after estimating the parameters) to those observed in the real data. The average absolute difference as a function of time interval length is shown in Fig. 5.2. We can see that the Aligned Bias User Response model is better than the Without Bias User Response model.

5.5.2 Reward

We use the correlation between exposures to fake and true news ([15]) to quantify our objective that people exposed more to fake news are also exposed more to true



(a) Twitter 2016

(b) Twitter 2015

Figure 5.2.: Difference in expected and observed number of likes

news. The number of exposures by time t is given by $\mathbf{G} \cdot \boldsymbol{\mathcal{N}}(t, h)$, and that in stage k can be obtained as $\mathbf{G} \cdot \boldsymbol{\mathcal{N}}(\tau_{k+1}, h) - \mathbf{G} \cdot \boldsymbol{\mathcal{N}}(\tau_k, h)$, i.e. $\mathbf{G} \cdot \mathbf{n}_k(h)$. Thus, the reward is given as,

$$R(\mathbf{s}_k) = \frac{1}{N}(\mathbf{n}_k(T))^\top \mathbf{G}^\top \mathbf{G}\, \mathbf{n}_k(F) \tag{5.7}$$

5.5.3 Policy Learning and Optimization

Regulating policy at different time steps helps model the dynamic behavior of people, for e.g., a user is active for a certain time period time and becomes inactive afterwards. Also, in our problem, we impose a budget constraint on the total amount of intensities allocated to users, and thus, regulating their distribution is important. Therefore, we consider multi-stage interventions, i.e., interventions at regular spanned time intervals.

Our goal is to learn policy π to determine the intervention to be applied at each stage for true news diffusion process such that the total expected discounted reward

for all stages, $J = \sum_{k=1}^{K} \gamma^k \mathbb{E}[R(\mathbf{s}_k, \mathbf{a}_k)]$ is maximized, where $\gamma \in (0, 1]$ is the discount rate.

In order for our policy to have long-term impact, we consider both immediate and future rewards. But, as a post ages, its influence decreases [148]. In social networks, the feeds are chronologically sorted [44], and thus, a user sees most recent posts from her peers than older. This indicates that reward from recent stages is more important than that from later stages in time. Since, our reward is based on number of exposures to a post, we use discounted rewards setting (as used in previous work (e.g., [15,31]), that is easier to realize using multi-stage interventions described above. Using Eq. 5.7, we can write,

$$\mathbb{E}[R_k(\mathbf{s}_k, \mathbf{a}_k)] = \frac{1}{N}\mathbb{E}[(\mathbf{n}_k(T))^\top \mathbf{G}^\top \mathbf{G}\ \mathbf{n}_k(F)] = \frac{1}{N}\mathbb{E}[\mathbf{n}_k(T)]^\top \mathbf{G}^\top \mathbf{G}\ \mathbb{E}[\mathbf{n}_k(F)] \quad (5.8)$$

The expected reward for fake and true news diffusion processes can be decomposed due to the independence assumption. Following [15] and [143], we obtain,

$$\mathbb{E}[\mathbf{n}_k(T)] = \mathbf{\Gamma}_h(\boldsymbol{\mu}_T + \mathbf{a}_{T,k}) + \mathbf{\Upsilon}_h \mathbf{y}_{T,k} \quad (5.9)$$

$$\mathbb{E}[\mathbf{n}_k(F)] = \mathbf{\Gamma}_h \boldsymbol{\mu}_F + \mathbf{\Upsilon}_h \mathbf{y}_{F,k} \quad (5.10)$$

where, $\mathbf{y}_{T,k}$ and $\mathbf{y}_{F,k}$ are defined in Sec. 5.4.2, that capture the effect of history due to past events, $\mathbf{\Upsilon}_h = (\mathbf{\Phi} - \omega_h \mathbf{I})^{-1}(e^{(\mathbf{\Phi}-\omega_h \mathbf{I})(\Delta)} - \mathbf{I})$ and $\mathbf{\Gamma}_h = \mathbf{\Upsilon}_h + (\mathbf{\Phi} - \omega_h \mathbf{I})^{-1}(\mathbf{\Upsilon}_h - \mathbf{I}(\Delta))/\omega_h$. Thus, we compute the expected reward as,

$$\begin{aligned}\mathbb{E}[R_k(\mathbf{s}_k, \mathbf{a}_k)] = \frac{1}{N}(\mathbf{\Gamma}_T(\boldsymbol{\mu}_T + \mathbf{a}_k) + \mathbf{\Upsilon}_T \mathbf{y}_{T,k})^\top \\ \mathbf{G}^\top \mathbf{G}\ (\mathbf{\Gamma}_F \boldsymbol{\mu}_F + \mathbf{\Upsilon}_F \mathbf{y}_{F,k})\end{aligned} \quad (5.11)$$

The linear dependence of expected reward on policy $\mathbf{a}_k$ results in a convex optimization problem. Similarly, we calculate $\mathbb{E}[\mathbf{w}_k]$ (as in [143]).

We represent the policy as a function of state ($\mathbf{s}_k$), parameterized by weights $\boldsymbol{\theta}$, that is, $\mathbf{a}_k = \pi(\mathbf{s}_k; \boldsymbol{\theta})$, where π is the function we want to learn. Each state is associated with a value $V(\mathbf{s}_k)$, that is, the total expected reward when in the given state following policy π, $V(\mathbf{s}_k) = \mathbb{E}[\sum_{j=k}^{K} \gamma^j R_j | (\mathbf{s}_k, \pi)]$. Since it is computationally expensive

to compute $V(\mathbf{s}_k)$ from future rewards for every possible policy π, we approximate the value as a function of the state parameterized by weights $\boldsymbol{\phi}$, that is, $V(\mathbf{s}_k) = f(\mathbf{s}_k; \boldsymbol{\phi})$, as in [30, 31]. Policy gradient methods are more effective in high dimensional spaces, and can learn continuous policies. Thus, we use state-of-the-art advantage actor-critic algorithm [28] to find the optimal policy. The details are presented in Algorithm 2.

Setup

Fig. 5.3 shows the complete training/test setup for our model. In the figure, $e[t, t']$ and $l[t, t']$ represent the post and like (feedback) events between time t and t'. We use MHP_λ to denote the MHP defined in Sec. 5.4.2, and MHP_ψ to denote the MHP defined in Sec. 5.5.1. We use the data from time $[0, K(= 10))$ to learn the parameters. Then we divide the remaining data corresponding to time interval $[K(= 10), 4K]$ into three parts, data from $[K, 2K)$ corresponds to *training dataset* used to learn the policy, data from $[2K, 3K)$ corresponds to *evaluation dataset* used to evaluate the learnt policy by measuring reward obtained, and data from $[3K, 4K]$ is used as *held-out dataset* for experiments in Sec. 5.6.2.

We obtain the training dataset and evaluation dataset by generating post (tweet) and feedback events using MHP_λ and MHP_ψ, respectively. Generating data using MHPs is supported by our observation that MHP_λ and MHP_ψ better capture the diffusion and feedback processes as shown in Sections 5.4.4 and 5.5.1. Moreover, since we cannot make real-time intervention to test the policy, we use a simulated environment (using MHPs) as a proxy for online interventions to measure the reward using evaluation data. In order to make the training and evaluation environment similar, we use the events generated by simulating MHPs with parameters learnt from real data. Moreover, we compute the expected value of reward in the future (next stage) assuming that the diffusion process follows the MHP.

Let there be K stages in the training data. Given features for stage k, we find the policy to be applied for stage $k + 1$. We use a multi-layer feed-forward neural

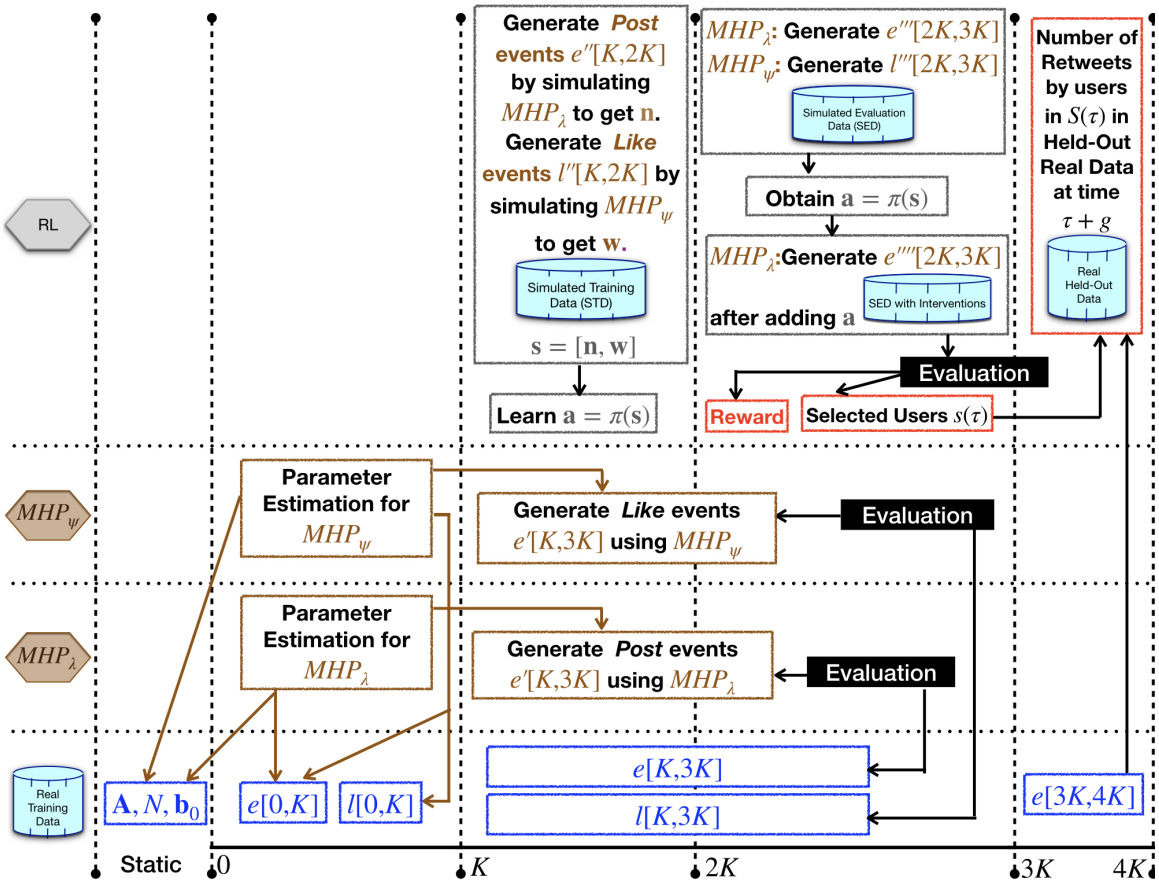


Figure 5.3.: Policy Learning and Evaluation Framework

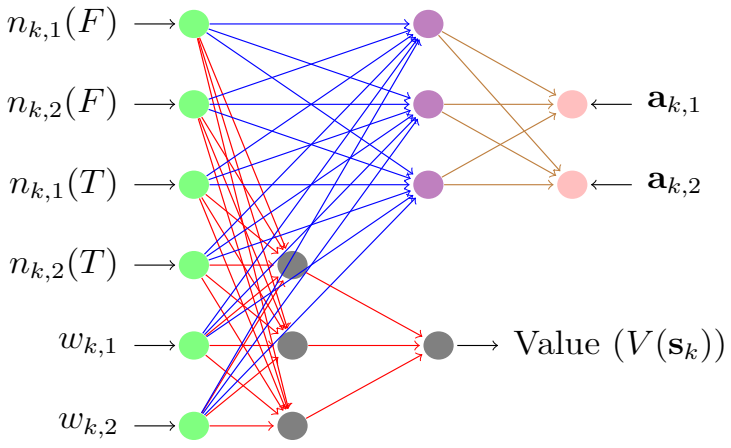


Figure 5.4.: Neural Network Architecture

Algorithm 2 Policy Learning and Optimization

Input: $\{\mathbf{s}_k\}_{k=1}^{K}, \boldsymbol{\mu}, \boldsymbol{\Phi}, \boldsymbol{\chi}, \omega, \gamma, \eta_\theta, \eta_\phi$
Output: $\boldsymbol{\theta}^*$
repeat
  for $k = 1, ..., K-1$ **do**
    $\mathbf{a}_{k+1} = \pi(\mathbf{s}_k; \boldsymbol{\theta})$
    $V(\mathbf{s}_k) = f(\mathbf{s}_k; \boldsymbol{\phi})$
    $\mathbf{a}_{k+1} = \frac{\mathbf{a}_{k+1}}{||\mathbf{a}_{k+1}||_1} \times O^k$ /* Budget Constraint */
    Compute $\mathbb{E}[R_{k+1}(\mathbf{a}_{k+1})]$ (Eq. 5.8)
    $\mathbf{n}_{k'}(h) = \mathbb{E}[\mathbf{n}_k(h)]$, $\mathbf{w}_{k'} = \mathbb{E}[\mathbf{w}_k], \mathbf{s}_{k'} = (\mathbf{n}_{k'}(h), \mathbf{w}_{k'})$
    $V(\mathbf{s}_{k'}) = f(\mathbf{s}_{k'}; \boldsymbol{\phi})$
    $r_k = \mathbb{E}[R_{k+1}(\mathbf{a}_{k+1})] + \gamma V(\mathbf{s}_{k'})$
  end for
  $L_\theta = 0, \quad L_\phi = 0$
  for $k = 1, ..., K$ **do**
    Let $D_k = \sum_{j=k}^{K} \gamma^{k-1} r_k$
    $B_k = D_k - V(\mathbf{s}_k)$ /* Compute Advantage */
    $L_\theta = L_\theta + B_k$
    $L_\phi = L_\phi + ||V(\mathbf{s}_k) - D_k||_2$
  end for
  $J_\theta = L_\theta, \quad J_\phi = -L_\phi$
  $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta_\theta \boldsymbol{\nabla}_\theta J_\theta, \quad \boldsymbol{\phi} = \boldsymbol{\phi} + \eta_\phi \boldsymbol{\nabla}_\phi J_\phi$
until $||\Delta\boldsymbol{\theta}|| < 0.1$
$\boldsymbol{\theta}^* = \boldsymbol{\theta}$
return $\boldsymbol{\theta}^*$

network (NN) to learn this policy π. The input to the NN, for stage k, is the state $\mathbf{s}_k = [\mathbf{n}_k(F), \mathbf{n}_k(T), \mathbf{w}_k]$. Hence, the dimensionality of input layer is $3N$, where N is the number of users in the network. Now, $V(\mathbf{s}_k) = f(\mathbf{s}_k; \boldsymbol{\phi})$, and $\mathbf{a}_{k+1} = \pi(\mathbf{s}_k; \boldsymbol{\theta})$. We

use the same NN to learn both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, however they are independent of each other. We use one hidden layer to learn the policy $\pi : \mathbf{s}_k \rightarrow \mathbf{a}_{k+1}$, and a separate hidden layer to learn the value function $V(\mathbf{s}_k)$. There are 2 different outputs of the NN, a scalar value $V(\mathbf{s}_k)$, and an N-dimensional output corresponding to the action $a_{k+1,i}$ for each user i. We used Adam optimizer and learning rate of 0.02. Fig. 5.4 shows the network for two users.

After we obtain the policy as output of the NN, we impose budget constraint by normalizing, as shown in line 7 in Alg. 2, and compute the expected reward for stage $k+1$ using $\mathbf{a}_{k+1}$ (line 8). $\mathbf{n}_{k'}$ and $\mathbf{w}_{k'}$ are the expected feature vector for the state $\mathbf{s}_{k'}$ obtained after applying the policy (line 9), and we find the expected value of the next state $V(\mathbf{s}_{k'})$ in line 10. r_k represents the expected reward that could be obtained by applying action $\mathbf{a}_{k+1}$ in state $\mathbf{s}_k$, in line 11, that comes from the Bellman Optimality Equation [151]. Instead of simply using the expected reward to optimize the policy, we use an advantage function that is obtained by subtracting the value of state as baseline from the expected reward. This helps to reduce the variance in estimates. Lines 15-16 show the computation of advantage function for each state $\mathbf{s}_k$. In lines 20-21, we learn the optimal parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, initialized randomly, using stochastic gradient descent, with learning rates η_θ and η_ϕ, respectively.

5.5.4 Policy Evaluation

To evaluate the learnt policy, we find the intervention $\mathbf{a} = \pi(\mathbf{s})$ where $s = (\mathbf{n}, \mathbf{w})$ obtained from events in the simulated evaluation data, as shown in Fig. 5.3. We simulate MHP_λ after adding $\mathbf{a}$ to the base exogenous intensity μ_T for true news diffusion, and compute the following evaluation metric.

Evaluation Metric To compare the performance of different methods, we consider the reward along with the fraction of users exposed to fake news that become exposed to true news. The latter helps to assign more importance to the mitigation of distinct users over mitigation of few users with high exposures. Let $L_{T,k}$ and $L_{F,k}$ be the sets

of users exposed to true and fake news, respectively, during stage k. $L_{T,k} = \{i | i \in [1, N], \mathbf{n}_k(T) \cdot \mathbf{G}_{.i} > 0\}$ and $L_{F,k} = \{i | i \in [1, N], \mathbf{n}_k(F) \cdot \mathbf{G}_{.i} > 0\}$. Define performance as $\mathcal{P} = \sum_{k=1}^{K} R_k \times \frac{|L_{T,k} \cap L_{F,k}|}{|L_{F,k}|}$ We measure the performance of a method relative to that obtained by applying no intervention, in order to assess the gains by making interventions. Specifically, we report the difference between the performance after applying the learnt policy and that without applying a policy. Note that since we cannot make real time interventions, we cannot explicitly test if there is a reduction in fake news spread.

5.6 Experiments

For experiments, we used $O_k \sim N \cdot \mathcal{U}(0, 1)$, and $\gamma = 0.7$. We compare our model, that we call MHP-U, against different baselines described below.

5.6.1 Baselines

Vanilla MHP (V-MHP)

Policy is a function of user events (tweets), similar to [15]), does not consider bias or feedback.

Exposure-based Policy (EXP)

To mitigate users who shared more posts related to fake news in past [15]: Intervention $a_{k,i} \propto \sum_{l=0}^{k} \sum_{j=1}^{N} G_{ij}\, n_{l,i}(F)$.

DEG

Intervention $a_i \propto$ degree of a user i (Sec. 5.4).

CEN

Intervention $a_i \propto$ closeness centrality of a user i (Sec. 5.4).

AVG

Intervention $a_{k,i} = \frac{O_k}{N}$, that is, the average budget per user.

Random (RND)

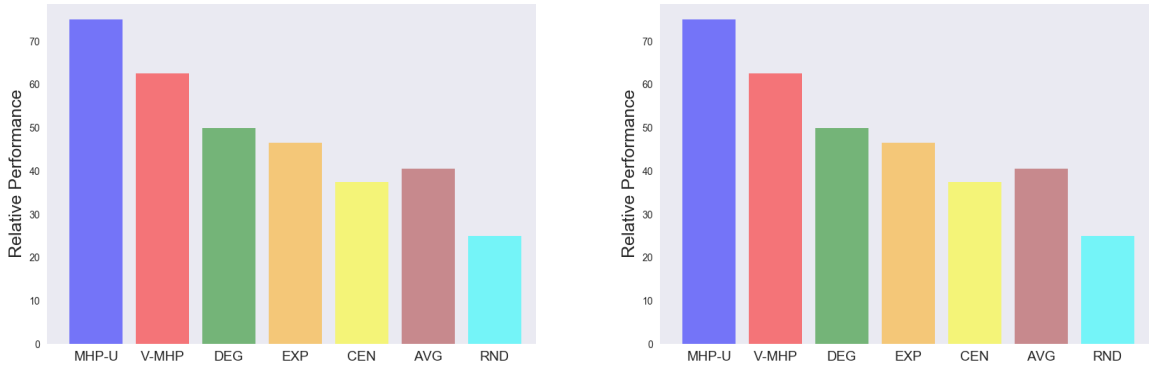
Intervention $a_{k,i} \propto \mathcal{U}(0, \frac{C_k}{N}]$

5.6.2 Results

Fig. 5.5 shows the relative performance of different methods. The correlation between fake and true news exposures is higher for MHP-U, and it also maximizes distinct number of users exposed to fake news. The gain in performance comes from the pairwise user feedback (MHP_ψ) modeled.

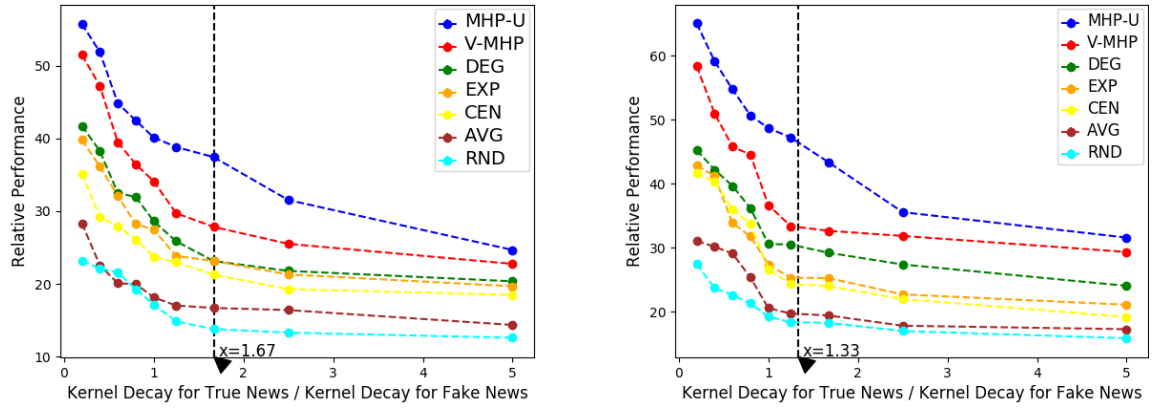
Fig. 5.6 shows a change in performance with respect to the ratio of decay parameter for true and fake news diffusion. As this ratio increases, the performance decreases exponentially. This is due to the fact that in the later stages, we have less true news events compared to fake news events. We mark the ratio corresponding to the settings described in Sec. 5.4.4, with a vertical line.

The above results serve as a proof of concept that providing incentives helps to increase the spread of true news even among the people exposed to fake news. However, since we cannot make any real-time interventions, we compare different methods by measuring the impact of nodes selected on held-out dataset (Sec. 5.5.3). Let $S(\tau)$ be the set of users who spread true news according to the model by time τ, that is, $S(\tau) = \{i | (\mathcal{N}_i(\tau, T) - \mathcal{N}_i(2K, T)) > 0\}$. We call these as *selected* users, and the remaining users are considered *missed* ($M(\tau)$) by the model. We consider $\tau \in [2K, 3K]$. Given users in $S(\tau)$ and $M(\tau)$, we calculate the total number of users



(a) Twitter 2016 (b) Twitter 2015

Figure 5.5.: Relative Performance on Twitter Datasets



(a) Twitter 2016 (b) Twitter 2015

Figure 5.6.: Relative Performance vs Ratio of Decay

who retweeted the posts of these users between time $[\tau^{'}, \tau^{'} + \Delta)$ where $\tau^{'} = \tau + g$, in order to measure the impact of the selected and missed nodes in terms of the people actually reached out in real data. $g = \{0, 2, 5, 8\}$ indicates the gap or number of stages after which we want to measure the impact (in the future). We considered different values of $\Delta \in \{1, 2, 3, 4, 5\}$ and report the average values in Table 5.1. We see that the impact of selected nodes (S) is greater than that of missed nodes (M) for MHP-U, and V-MHP by a large margin.

Table 5.1.: Sum of Retweets at $\tau + g$ for Users Selected at τ

MODEL	$\tau' = \tau + 0$		$\tau' = \tau + 2$		$\tau' = \tau + 5$		$\tau' = \tau + 8$	
	S	M	S	M	S	M	S	M
MHP-U	**1000.5**	**400.8**	**830.2**	**223.3**	**630.8**	170	**553.3**	**140.4**
V-MHP	700.4	416.6	553.7	250.4	420.9	**162.2**	369.1	148.8
DEG	590.6	500.3	445.3	350.1	330.3	259.8	296.6	233.4
EXP	600.2	575.7	299.1	437.8	210.5	298.8	200.2	291.7
CEN	538.5	569.9	369.2	334.6	283.7	257.6	246.1	223.3
AVG	420.7	598.2	260.3	449.1	215.4	325.6	173.3	300.2
RND	410.1	518.4	205.7	459.2	178.7	340.2	136.1	306.4

5.6.3 Experiments on Semi-Synthetic Data

We use subsets of twitter data to study the performance with respect to different network parameters. The results, on Twitter 2016, are shown in Fig. 5.7, where we highlight the region closely representing real-world scenarios. We observe that our method outperforms the baselines by larger margin in all such regions.

Ratio of out-degree to in-degree

Fake news sources have lower out-degree and high in-degree, compared to sources for true news [155, 156]. Fig. 5.7-a shows that the performance decreases as the ratio of ratio of out-degree to in-degree for sources of fake news to that of true news, increases. This can be due to a decrease in the number of followers for true news sources. This ratio is usually less than 1 in real networks [157].

Average Degree

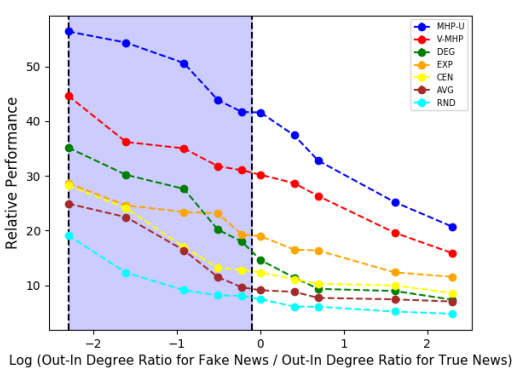
[158] showed that the average degree of users in fake news network (users who retweet fake news) is more than that in true news network. Fig. 5.7-b shows that the performance decreases as the ratio of average degree for fake news network to true news network increases, since fewer people are reached out by true news spreaders.

Centrality

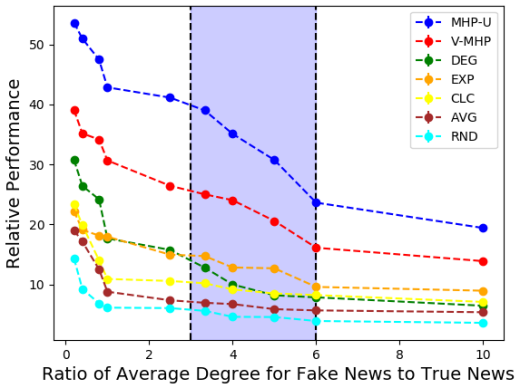
Closeness centrality for sources of fake news is higher than that for true news [4, 157]. Fig. 5.7-c shows change in performance with respect to ratio of average closeness centrality of fake news sources to that for true news. The performance is high for ratio 1.

Political Bias

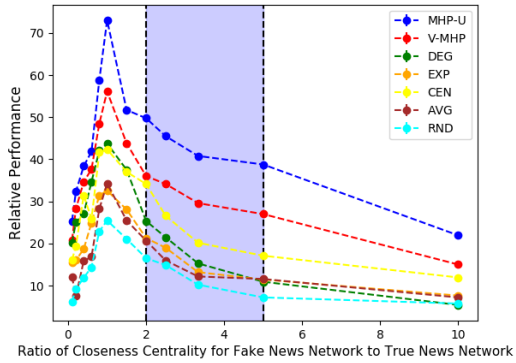
[1] observed that Democrats (D) are more likely to believe fake news than Republicans (R), and R are more likely to believe true news than D. We say that a user is High D (High R) if $b_{D,0,i} > 0.5$ ($b_{R,0,i} > 0.5$) and Low D (Low R) otherwise. Based on this, we create four groups of people. Group 1: High D fake, High R true news sources. Group 2: High D fake, Low R true news sources. Group 3: Low D fake, High R true news sources. Group 4: Low D fake, Low R true news sources. Fig. 5.7-d shows that the performance is least for Group 1, when bias is high for both R and D, indicating that it is difficult to encourage people to spread news that does not align with their ideology.



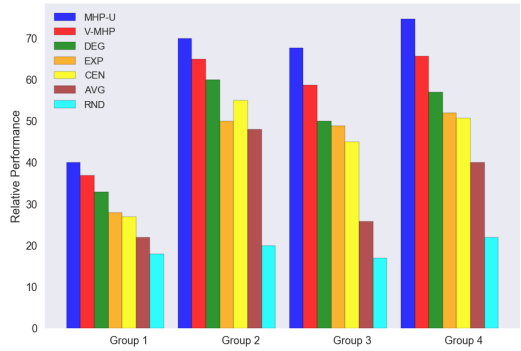
(a) Ratio of Out-to-in Degree



(b) Ratio of Average Degree



(c) Closeness Centrality



(d) Political Bias

Figure 5.7.: Relative Performance (Different Network Properties)

6 EFFICIENT CENTRALIZED SOCIAL REINFORCEMENT LEARNING

6.1 Introduction

News dissemination on Twitter is aided by two types of events—tweets, in which users share their own posts, and retweets, in which users re-post the information shared (tweeted) by others. Notably, as more people spread some news, others also start circulating it, irrespective of the underlying evidence, which is referred to as the *BandWagon* effect [159]. Thus, if more people spread true news, their peers will also share true news more, and social media sites can be a platform for people to help reduce the spread of fake news [160].

Intervention-based methods aim to mitigate the impact of fake news by increasing spread of true news [15]. These works conjecture that an increased exposure to true news will increase suspicion and mistrust for fake news, leading to a potential decrease in fake news spread. In the previous chapter, we proposed a centralized social reinforcement learning approach that learns how to incentivize users to share true news such that people exposed more to fake news also become exposed more to true news [43]. Multi-agent RL in a social context is difficult when there are a large number of users with relatively few interactions due to high dimensionality of the continuous state and action spaces. To reduce the computational cost, we previously considered tweets and retweets to be equivalent events.

However, some recent studies have observed large differences in the diffusion patterns of tweets and retweets, especially for fake news (e.g., [161]). They observed that the number of retweets for fake news are much higher than that for true news. And, retweets play a major role in the widespread and faster diffusion of news, compared to source tweets [3]. [162] found that Twitter has a low level of reciprocity and the influence between pair of users is asymmetric. They also observed that tweets are

able to reach a greater audience due to the retweets. The retweet network is very different from the followers network, and most of the retweets are from users who do not follow the source user of the tweet [3, 163]. [158] studied the differences between the network structure of fake and true news diffusion, and found that on an average, users in the fake news diffusion network retweet more and are retweeted more, compared to those in the true news diffusion network. To capture these differences, we model the tweet and retweet events with separate processes, which increases the model dimensionality, particularly when the goal is to learn centralized policies and decentralized learning cannot scale to large number of users.

In the previous chapter ([43]), we presented a Social RL approach to mitigate the impact of fake news by learning interventions to increase the intensity of true news diffusion. We employed joint policy learning to capture agent dependencies, however, we still estimate $\sim N^2$ parameters resulting in high computational cost and larger variance in policy estimates due to sparse interactions. Also, due to the high-dimensionality of joint state/action representation in the previous approach, the impact of an individual's state features in learning her own policy diminishes with larger N, resulting in noisy estimates as the policy parameters overfit to a single type of users who may be in the majority. This also makes it infeasible to model *reciprocity* in agent interactions, because N^2 agent *pairs* would lead to $N \times N^2 = N^3$ parameters. To avoid this, previously we only considered individual (i.e. N) actions per user.

To overcome these challenges, we propose a *Dynamic Cluster-based Policy Learning* (DCPL) approach to Social RL that utilizes the properties of the social network structure and agent correlations to obtain a compact model to represent the system dynamics. Specifically, we propose to cluster similar users in order to reduce the effective number of policies to be learned, and overcome the problem of sparse data by aggregating the interactions of similar users. We then develop a method to easily derive personalized agent-level actions from cluster-level policies by exploiting variability in agents' behaviors. Thus, we reduce the problem of learning policies for N users to that for $\mathcal{C}$ clusters ($\mathcal{C} \ll N$), and hence, the model dimensionality from $\sim N^3$

(with pairwise agent interactions) to $\sim\mathcal{C}^3$, to allow efficient and effective *joint* policy learning, considering all users. To consider individual contributions to the global network reward, we design clustering features motivated by *difference reward* [35]. Specifically, we define *contribution* features to measure a user's efficacy given other agents' actions, and, *payoff* features to determine how responsive an agent is to the policy applied in the past. We use these features (via clusters) to learn agents' effectiveness early on, for better exploring the action space, without increasing the state space, in order to speed-up convergence. Since agent interactions are dynamic, we update the policy after regular time-intervals and dynamically align and update the cluster memberships to reflect the effects of applying the updated policy. To the best of our knowledge, DCPL is the first MARL method to consider policy learning while dynamically clustering users, in social networks.

We evaluate the performance of DPCL compared to different static clustering and non-clustering policy learning methods, using real-world Twitter datasets. Results show that it is important to model the tweet and retweet events as separate processes. Compared to other baselines, our dynamic cluster-based approach is able to learn better policies that achieve higher *network reward*, by learning a compact model that reduces the effective number of policies required to be estimated.

6.2 Related Work

[15] proposed to mitigate the impact of fake news by increasing the diffusion of true news that is characterized using Multivariate Hawkes Process (MHP). [43] developed a Social RL approach to combat fake news spread by providing users incentives to share more true news, that are learnt using history of user events, and feedback. Since the method in [43] outperforms [15], we do not compare our proposed method against the latter.

[44] learned a separate model for each agent. However, this ignores inter-agent dependencies, and is infeasible for large N. [15, 43] developed centralized learning

approaches for fake news mitigation on Twitter, by modeling agent interactions via tweets and *likes*. They consider individual (i.e. N) actions and estimate $\sim N^2$ parameters resulting in high computational cost for large networks. Also, they do not address the problem of sparse interactions that leads to increased variance in estimates. In contrast, we reduce the model dimensionality by clustering similar users. This allows us to efficiently learn a more compact model and facilitates modeling actions per user *pair* (i.e. N^2 actions) to capture reciprocity in user interactions. Moreover, it addresses the problem of sparse interaction data by aggregating the interactions of similar users.

[15, 43] did not distinguish between tweets and retweets, and learned the same action for both activities. However, the diffusion patterns of tweets and retweets have large differences, especially, for fake news [161]. Moreover, retweet network is very different from followers network on Twitter with a low level of reciprocity [3]. To capture these differences, we consider tweet and retweet as separate network activities, resulting in three different types of agent interactions. Also, we learn actions per user pair to capture the reciprocity in user interactions, i.e., N^2 actions. However, this leads to an increase in the model dimensionality, particularly for joint learning, resulting in large number of parameters ($\sim N^3$), and high computational cost.

To overcome this, and learn better resource allocation strategies, we propose to first cluster users with latent features (based on their past behavior and contribution to the objective). To reduce the computational complexity, previously [74] considered an aggregate of agents to obtain a smaller effective number of agents. However, their approach is applicable to only small restricted discrete action spaces [97]. [72,73] used an aggregate statistic of agents' actions along with certain strong assumptions on the state-transition model. However, Social RL problems usually have continuous spaces describing the intricate user interactions and network activities, and the complex state-transition dynamics are generally unknown.

Previous work utilized *symmetry* between states and/or agents to reduce the size of the Markov Decision Process (e.g., [53, 54]). Our clustering approach is based on

the observation that similar users tend to show similar behavior, and thus we propose to learn similar policies for them. This has also been utilized to cluster users in mobile health domain [164,165]. However, they did not consider user interactions and dependencies, clustered users only once, and assumed the cluster assignments remain fixed (*static*). In our setting, we consider dynamic user interactions. Thus, there is a need to dynamically update cluster assignments, and we propose an approach to ensure cluster alignment at different time-steps.

6.3 Problem Definition

We consider a social network with N users who interact via d different network activities (e.g. tweet, like). Each user $i \in \{1, ..., N\}$ is an agent. Let $\mathbf{G}$ represent the followers adjacency matrix for the social network graph, where $G_{ij} = 1$ if j follows i, and 0 otherwise. Our data contains a temporal stream of events with the time horizon $[0, \mathcal{T})$ divided into K stages, each of time-interval Δ, where stage $k \in [1, K]$ corresponds to the time-interval $[\tau_k, \tau_{k+1})$. We consider three types of events characterizing user activities, corresponding to tweets ($\mathscr{T}$), retweets ($\mathscr{R}$), and likes ($\mathscr{L}$). We represent the tweet or retweet events using $e = (t, i, h, z)$ where t is the time-stamp at which user i shares a post of type $z = \mathscr{T}$ or $\mathscr{R}$, with label $h = F$ (Fake) or T (True). Like events are represented as $l(u, i, t)$ indicating user i likes user u's post at time t.

A quantitative measure of the impact of fake and true news is the number of people exposed. Let $\mathcal{N}_i(t, h, z)$ represent the number of times user i shares news of type z = $\mathscr{T}$ or $\mathscr{R}$, with label $h = F$ or T, up to time t. Then the number of times a user i is exposed to news up to time t corresponds to $G_{.i} \cdot \mathcal{N}(t, h, z)$. Let $W_i(t)$ be the number of likes received by user i upto time t. Our goal is to incentivize users to share true news—to ensure that users receive at least as much true as fake news. Algorithmically, we want to increase the probability of sharing true news in a targeted fashion by learning an efficient strategy to allocate incentive among users, based on

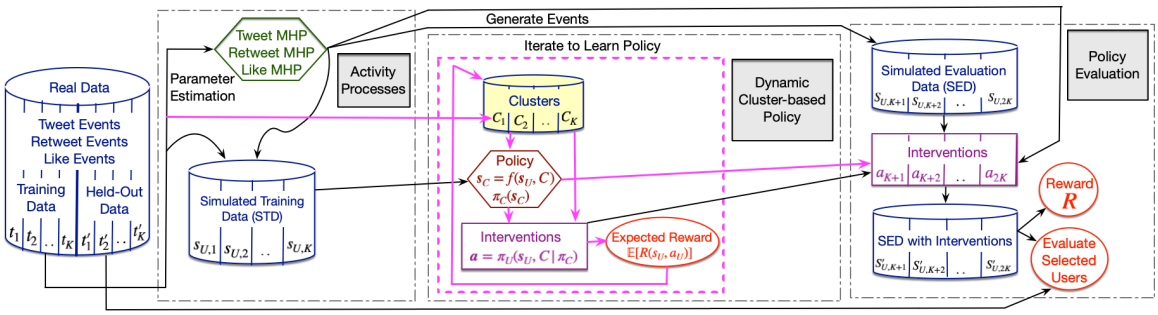


Figure 6.1.: Overview of Cluster-Based Policy Learning and Evaluation

properties of the network $\mathbf{G}$ and the effects of past user interactions. Specifically, given the network *state* $\mathbf{s} \in \mathbb{R}^{dN}$ corresponding to d user activities (described further in Sec. 6.4.3), we want to learn an incentivization policy $\pi : \mathbf{s} \to \mathbf{a}$ to obtain incentive *actions* $\mathbf{a} \in \mathbb{R}^{N+N^2}(\mathbf{a} \geq 0)$, where tweet actions are per user (i.e., N) and retweet actions are per user pair (i.e., N^2). Incentive corresponds to increasing the likelihood of users sharing true news, and can be realized (in real-world) by using external motivation (e.g., money, reward/credit points). Note that fake news mitigation is a multifaceted problem, and the reward function depends on the application under consideration. In this work, we consider *reward* as the correlation between exposures to fake and true news, based on the idea that users exposed more to true news must also be exposed more to fake news so that they are less likely to believe in fake news in the future [15].

6.4 Cluster-Based Social Reinforcement Learning Approach

6.4.1 Overview

Fig. 6.1 illustrates the different components of our system. Our key insight is to decouple the processes governing tweet, retweet and like events. This helps to study the effect of different types of events in news diffusion, and learn an *approximate* model more efficiently than full joint learning. We map the excitation events to states in

a Markov Decision Process (MDP), and learn interventions to increase the intensity function for true news diffusion. To further increase efficiency, we cluster similar users together so that we can learn a smaller *cluster-level* intervention policy, from which we can easily derive individual interventions for the members of each cluster.

Specifically, from the training data we learn a set of Multivariate Hawkes Processes (MHP), one for each type of activity event, and also cluster the users. We use the MHP models to simulate additional training data for learning the policy. While learning the policy function π_C, we compute state features for each cluster (based on its current members) and compute the interventions for clusters given the current policy, and derive users interventions. Then we calculate expected reward and use this to further optimize the policy and update the cluster memberships.

To evaluate the estimated policy $\hat{\pi}_C$, we simulate data again from the MHPs. Using the final clusters C_K, we obtain interventions from the policy to add to the MHP intensity functions and generate evaluation data to assess empirical reward. We also assess the effectiveness of the users selected by our model to promote true news by measuring their number of retweets in held out training data. Each component is described in more detail next.

6.4.2 Activity Processes

We use N-dimensional MHPs ([37]) to model user activities and simulate network dynamics. Since we cannot make real-time intervention to test the policy, we require a simulated environment as a proxy for online interventions to measure the reward using evaluation data. [43] illustrated that MHPs closely model news diffusion and feedback processes in real-world, and thus, we use the events generated by simulating MHPs with parameters learned from real data. Let $\lambda_{h,z,i}$ be the intensity function governing the sharing rate of user i, where $h = F$ or T, and $z = \mathscr{T}$ or $\mathscr{R}$:

$$\lambda_{h,z,i}(t) = \mu_{h,z,i} + \sum\nolimits_{j=1}^{N} \int_0^t \Phi_{z,ji}\ (\omega_{h,z} e^{-\omega_{h,z} t})\ d\mathcal{N}_j(s, h, z)$$

where, the integral is over time, and s is used as placeholder for limits {0,t}. $\mu_{h,z,i}$ is the base exogenous intensity of user i, $\mathbf{\Phi}_z$ is the kernel adjacency matrix estimated from the training data, and $\omega_{h,z}e^{-\omega_{h,z}t}$ is the exponential Hawkes kernel. For **Tweet MHP** and **Like MHP**, we use MHP models proposed in Chapter 5 to obtain $\mathcal{N}_i(t, h, \mathscr{T})$ and $W_i(t)$. For the **Retweet MHP**, we need to estimate the (asymmetric) influence between all pair of users, i.e., $\Phi_{\mathscr{R},ji}$ to capture reciprocity. This naively requires $\sim N^2$ parameters, but we use a low-rank approximation of the kernel matrix, proposed in [166], to improve efficiency. Our policy will learn actions that correspond to *interventions* to increase sharing of true news events ($h = T$) only. Let $a_{z,k,i}$ be a constant intervention action for user i at stage k (time $t \in [\tau_k, \tau_{k+1})$), added to the her base intensity.

$$\lambda_{T,z,i}(t) = \mu_{T,z,i} + a_{z,k,i} + \sum_{j=1}^{N} \int_0^t \Phi_{z,ji} \ (\omega_{T,z}e^{-\omega_{T,z}t}) \ d\mathcal{N}_j(s, T, z)$$

To simulate the event data, we interleave the MHPs to generate tweet events, then retweet events, and then like events. This ensures that the training and evaluation environments are similar.

6.4.3 Dynamic Cluster-based Policy

Our goal is to learn a policy π that maps the state representation (over N users) to intervention actions for tweet and retweet intensities ($N + N^2$ actions). To lower the computational cost, our key insight is to utilize agent correlations to reduce the size of the MDP. Specifically, we propose to cluster users into $\mathcal{C}$ clusters, so that we can learn a policy π_C that maps the state representation of $\mathcal{C}$ clusters to $\mathcal{C} + \mathcal{C}^2$ actions, where $\mathcal{C} \ll N$. We use a fixed number of clusters because we assume that the set of user *types* don't change much over time, but we allow users to move from cluster to cluster. We then develop a method to derive user-level actions from the cluster-level actions. Let $c_{k,i}$ be the cluster of user i at stage k, where $c_{k,i} = m$ ($m \in [1, \mathcal{C}]$), and $C_k = \{c_{k,i}\}_{i=1}^N$ is the set of all cluster assignments at stage k. We define the cluster

membership matrix, $\boldsymbol{\mathcal{M}}_k \in \{0,1\}^{N\times\mathcal{C}}$, at stage k, such that $\mathcal{M}_{k,i,m}=1$ if $m=c_{k,i}$, and 0 otherwise.

Alg. 3 outlines our approach to learning a cluster-based policy π_C parameterized by $\boldsymbol{\theta}$, given simulated training data ($\{\mathbf{s}_{U,k}\}_{k=1}^{K}$), initial cluster memberships ($\boldsymbol{\mathcal{M}}_1$), and user features ($\mathbf{X}_1$) as input, with hyperparameters $\gamma, \eta_\theta, \eta_\phi, \delta$. We first compute the state features of the clusters by averaging the state features of their associated members. Then, given the state features per cluster, we apply the current policy to obtain cluster-level actions $\mathbf{a}_{C,z,k}$. Next, we compute the cluster centroids and derive user-level actions $\mathbf{a}_{U,z,k}$, based on the distance of the user to the centroids (see Alg. 4). Then we compute the expected reward based on the user-level actions (Alg. 5), and calculate user *payoff* and *contribution* features $\mathbf{X}$ (described later) to recluster users into $\mathcal{C}$ clusters (see Alg. 7). Finally, we update the policy parameters $\boldsymbol{\theta}$ by first computing the objective (see Alg. 6) based on the expected reward, and then using stochastic gradient descent with learning rates η_θ and η_ϕ. The algorithm repeats until convergence and returns the final policy parameters and cluster memberships. We describe each component next.

State Features

We represent the network state $\mathbf{s}_k \in \mathbb{R}^{dN}$ at stage k as the number of events for d different network activities (i.e., interactions) in the previous stage (e.g., in [15, 152, 153]). Specifically, $s_{k,i,j}$ is the number of events for the $j^{th}(j\in[1,d])$ activity that user $i\in[1,N]$ has performed in stage $k-1$. Let $n_{k,i}(h,z)=\mathcal{N}_i(\tau_k,h,z)-\mathcal{N}_i(\tau_{k-1},h,z)$ represent the number of times user i shares news in stage $k-1$, and $w_{k,i}=W_i(\tau_k)-W_i(\tau_{k-1})$ represent the number of likes received by user i. Let $\mathbf{s}_{U,k,i}=(n_{k,i}(T,\mathscr{T}), n_{k,i}(F,\mathscr{T}), n_{k,i}(T,\mathscr{R}), n_{k,i}(F,\mathscr{R}), w_{k,i})$ be the state feature for user i that is input to Alg. 3. We compute the state features for cluster m, at stage k, as the mean of the state features of its members (line 5, Alg. 3). Thus, there are $d=5$ network activities

Algorithm 3 Dynamic Cluster-based Policy Optimization

Input: $\{\mathbf{s}_{U,k}\}_{k=1}^{K}, \boldsymbol{\mathcal{M}}_1, \mathbf{X}_1, \gamma, \eta_\theta, \eta_\phi, \delta$
repeat
  for $k = 1, ..., K$ **do**
    /* Compute cluster state features */
    $\mathbf{s}_{C,k,m}$=$\sum_{i=1}^{N} \mathcal{M}_{k,i,m}\mathbf{s}_{U,k,i} / \sum_{i=1}^{N} \mathcal{M}_{k,i,m}$, $\forall m$
    $\mathbf{a}_{C,z,k} = \boldsymbol{\pi}_C(\mathbf{s}_{C,k}; \boldsymbol{\theta})$ /* Get cluster-level actions */
    /* Compute cluster centroids */
    $\mathbf{Y}_{k,m} = \sum_{i=1}^{N} \mathcal{M}_{k,i,m}\mathbf{X}_{k,i} / \sum_{i=1}^{N} \mathcal{M}_{k,i,m}$, $\forall m$
    /* Obtain user-level actions with Alg. 4 (π_U) */
    $\mathbf{a}_{U,z,k} = \pi_U(\mathbf{a}_{C,z,k}, \mathbf{X}_k, \mathbf{Y}_k)$
    r_k = GetExpectedReward$(\mathbf{s}_{U,k}, \mathbf{a}_{U,z,k}, \boldsymbol{\phi}, \gamma)$
    Obtain $\mathbf{X}_{k+1}$ using $\mathbf{a}_{U,z,k}, \mathbf{a}_{U,z,k-1}$ /* Eq. 6.10, 6.11 */
    $\boldsymbol{\mathcal{M}}_{k+1}$ = UpdateClusters$(\mathbf{Y}_k, \boldsymbol{\mathcal{M}}_k, \mathbf{X}_{k+1}, \delta)$
  end for
  /* Learn Policy and update parameters */
  J_θ, J_ϕ = GetTotalObjective$(\{r_k\}_{k=1}^{K}, \{\mathbf{s}_{U,k}\}_{k=1}^{K}, \boldsymbol{\phi}, \gamma)$
  $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta_\theta \boldsymbol{\nabla}_{\boldsymbol{\theta}} J_\theta, \quad \boldsymbol{\phi} = \boldsymbol{\phi} + \eta_\phi \boldsymbol{\nabla}_{\boldsymbol{\phi}} J_\phi$
until $|\Delta\boldsymbol{\theta}| < \delta$ /* Convergence */
$\boldsymbol{\theta}^* = \boldsymbol{\theta}, \boldsymbol{\mathcal{M}}^* = \boldsymbol{\mathcal{M}}_K$
return $\boldsymbol{\mathcal{M}}^*, \boldsymbol{\theta}^*$

corresponding to tweets (T/F), retweets (T/F), and likes, and the dimensionality of the state representation is $5N$, which will be reduced to $5\mathcal{C}$ once we cluster users.

Reward

We consider the reward function to be the correlation between exposures to fake and true news. The number of exposures by time t is given by $\mathbf{G} \cdot \boldsymbol{\mathcal{N}}(t, h, z)$, and

that in stage k can be obtained as $\mathbf{G} \cdot \boldsymbol{\mathcal{N}}(\tau_{k+1}, h, z) - \mathbf{G} \cdot \boldsymbol{\mathcal{N}}(\tau_k, h, z)$, i.e. $\mathbf{G} \cdot \mathbf{n}_k(h, z)$. Thus, the reward is given as,

$$R_z(\mathbf{s}_{U,k}) = \frac{1}{N}(\mathbf{n}_k(T, z))^\top \mathbf{G}^\top \mathbf{G}\, \mathbf{n}_k(F, z) \tag{6.1}$$

Since, retweets are prominent in news diffusion compared to tweets (Sec. 6.1), we consider the cumulative reward R as,

$$R(\mathbf{s}_{U,k}) = \zeta\, R_{\mathscr{T}}(\mathbf{s}_{U,k}) + R_{\mathscr{R}}(\mathbf{s}_{U,k}) \tag{6.2}$$

where ζ is the relative importance of reward from tweet events compared to that from retweet events.

Objective

Our goal is to learn a cluster-based policy π_C to determine the interventions to be applied to users, at each stage, for true news diffusion process such that the total expected discounted reward for all stages, $J = \sum_{k=1}^{K} \gamma^k \mathbb{E}[R(\mathbf{s}_{U,k}, \mathbf{a}_{U,\mathscr{T},k}, \mathbf{a}_{U,\mathscr{R},k})]$ is maximized, where $\gamma \in (0, 1]$ is the discount rate. We map the interventions to actions in MDP. We impose a budget constraint on the total amount of intervention that can be applied to all users i.e. $||\mathbf{a}_{U,z,k}||_1 = O_{z,k}$, where $O_{z,k}$ is the total budget at stage k (line 8, Alg. 4). Using Eq. 6.1, we can write,

$$\mathbb{E}[R_z(\mathbf{s}_{U,k}, \mathbf{a}_{U,z,k})] = \frac{1}{N}\mathbb{E}[\mathbf{n}_k(T, z)]^\top\, \mathbf{G}^\top \mathbf{G}\;\; \mathbb{E}[\mathbf{n}_k(F, z)]$$

We assume that the diffusion of fake and true news is independent, and thus, decompose the expected reward. $\mathbb{E}[\mathbf{n}_k(h, z)]$. The cumulative expected reward due to tweets and retweets is, $\mathbb{E}[R(\mathbf{s}_{U,k}, \mathbf{a}_{U,\mathscr{T},k}, \mathbf{a}_{U,\mathscr{R},k})] = \mathbb{E}[R_{\mathscr{T}}(\mathbf{s}_{U,k}, \mathbf{a}_{U,\mathscr{T},k})] + \mathbb{E}[R_{\mathscr{R}}(\mathbf{s}_{U,k}, \mathbf{a}_{U,\mathscr{R},k})]$. The computation of $\mathbb{E}[\mathbf{n}_k(h, z)]$ is similar to that in Sec. 5.5.3. Specifically,

$$\mathbb{E}[\mathbf{n}_k(T, z)] = \boldsymbol{\Gamma}_{h,z}(\boldsymbol{\mu}_{T,z} + \mathbf{a}_{T,z,k}) + \boldsymbol{\Upsilon}_{h,z}\mathbf{y}_{T,z,k} \tag{6.3}$$

$$\mathbb{E}[\mathbf{n}_k(F, z)] = \boldsymbol{\Gamma}_{h,z}\boldsymbol{\mu}_{F,z} + \boldsymbol{\Upsilon}_{h,z}\mathbf{y}_{F,z,k} \tag{6.4}$$

where, $\mathbf{y}_{T,z,k}$ and $\mathbf{y}_{F,z,k}$ are as defined in Section 5.4.2, that capture the effect of history due to past events, and, $\boldsymbol{\Upsilon}_{c,z} = (\boldsymbol{\Phi}_z - \omega_{c,z}\mathbf{I})^{-1}(e^{(\boldsymbol{\Phi}_z - \omega_{c,z}\mathbf{I})(\Delta)} - \mathbf{I})$, $\boldsymbol{\Gamma}_{c,z} = \boldsymbol{\Upsilon}_{c,z} + (\boldsymbol{\Phi}_z - \omega_{c,z}\mathbf{I})^{-1}(\boldsymbol{\Upsilon}_{c,z} - \mathbf{I}(\Delta))/\omega_{c,z}$. Thus, we compute the expected reward as,

$$\begin{aligned}\mathbb{E}[R_{z,k}(\mathbf{s}_{U,k}, \mathbf{a}_{z,k})] = \frac{1}{N}(\boldsymbol{\Gamma}_{T,z}(\boldsymbol{\mu}_{T,z} + \mathbf{a}_{z,k}) + \boldsymbol{\Upsilon}_{T,z}\mathbf{y}_{T,z,k})^\top \\ \mathbf{G}^\top\mathbf{G}\ (\boldsymbol{\Gamma}_{F,z}\boldsymbol{\mu}_{F,z} + \boldsymbol{\Upsilon}_{F,z}\mathbf{y}_{F,z,k})\end{aligned} \tag{6.5}$$

Similarly, we calculate $\mathbb{E}[\mathbf{w}_k]$ (refer [143] for more details). Using Eq. 6.2, we obtain,

$$\mathbb{E}[R(\mathbf{s}_{U,k}, \mathbf{a}_{\mathcal{T},k}, \mathbf{a}_{\Re,k})] = \zeta\ \mathbb{E}[R_{\mathcal{T}}(\mathbf{s}_{U,k}, \mathbf{a}_{\mathcal{T},k})] +\ \mathbb{E}[R_{\Re}(\mathbf{s}_{U,k}, \mathbf{a}_{\Re,k})] \tag{6.6}$$

From Equations 6.3, 6.4 and 6.5, we can write the expected number of events and reward for a user as follows,

$$\mathbb{E}[n_{k,i}(T,z)] = \mathbb{1}^\top\boldsymbol{\Gamma}_{T,z,i,.}(\mu_{T,z,i} + a_{T,z,i}) + \mathbb{1}^\top\boldsymbol{\Upsilon}_{T,z,i,.}y_{T,z,k,i} \tag{6.7}$$

$$\mathbb{E}[n_{k,i}(F,z)] = \mathbb{1}^\top\boldsymbol{\Gamma}_{F,z,i,.}\mu_{F,z,i} + \mathbb{1}^\top\boldsymbol{\Upsilon}_{F,z,i,.}y_{F,z,k,i} \tag{6.8}$$

$$\begin{aligned}\mathbb{E}_i[R_{z,k}(\mathbf{s}_{U,k,i}, \mathbf{a}_{z,k,i})] &= \frac{1}{N}\mathbb{E}_i[(n_{k,i}(T,z)\ n_{k,i}(F,z))\ \mathbf{G}_{.i}^\top\mathbf{G}_{.i}] \\ &= \frac{1}{N}\mathbb{E}_i[n_{k,i}(T,z)]\ \mathbb{E}_i[n_{k,i}(F,z)]]\ \mathbf{G}_{.i}^\top\mathbf{G}_{.i}\end{aligned} \tag{6.9}$$

We can substitute Equations 6.7 and 6.8 in Equation 6.9 to compute the expected reward for each user i.

Clustering Features

We design clustering features based on *Difference Reward* (DR) that use a user's contribution to *shape* the reward signal and reduce noise in policy estimates. We define *payoff* features that indicate how responsive a user is to the policy applied in the past, by measuring the change in a user's expected reward after applying policy in stage $k-1$.

$$p_{z,k,i} = \mathbb{E}_i[R_z(\mathbf{s}_{U,k-1}, \mathbf{a}_{U,z,k-1})] - \mathbb{E}_i[R_z(\mathbf{s}_{U,k-2}, \mathbf{a}_{U,z,k-2})] \tag{6.10}$$

We define *contribution* features $q_{z,k,i}$ to measure user i's contribution in the expected reward, given the actions of other users. Specifically, we calculate the difference in the total expected reward obtained on providing incentive to the user, to when no incentive is provided to the user. To compute the latter, we set the incentive for user i as 0.

$$\begin{aligned} q_{z,k,i} = \mathbb{E}_i[R_z(\mathbf{s}_{U,k-1}, \mathbf{a}_{U,z,k-1})] \\ - \mathbb{E}_i[R_z(\mathbf{s}_{U,k-1}, (\{a_{U,z,k,j}\}_{j=1,j\neq i}^{N}; a_{U,z,k-1,i} = 0))] \end{aligned} \tag{6.11}$$

Thus, the complete set of features used for clustering users at the start of stage k is $\mathbf{X}_{k,i} = (p_{\mathscr{T},k,i}, p_{\mathscr{R},k,i}, q_{\mathscr{T},k,i}, q_{\mathscr{R},k,i}), i \in [1, N]$. The similarity between two users i and j is based on the Euclidean distance between their respective feature vectors $\mathbf{X}_{k,i}$ and $\mathbf{X}_{k,j}$. We do not know the policy estimates apriori for the first stage, and obtain initial clusters C_1 and membership $\boldsymbol{\mathcal{M}}_1$, using K-means++, based on empirical rewards computed from training data. This captures the *natural policy* or intrinsic behavior of users to spread news, without external incentives. We incorporate the DR signals as input to the policy function approximator (via clustering features), rather than using them as explicit shaped reward signal that is different for each agent. This helps to avoid learning a separate model for each user as in the standard DR shaping techniques (e.g., [35]).

Learning Cluster-based Policy

Given cluster-level state features $\mathbf{s}_{C,k}$, our goal is to learn a cluster-based policy parameterized by $\boldsymbol{\theta}$, i.e., $\mathbf{a}_{C,\mathscr{T},k}, \mathbf{a}_{C,\mathscr{R},k} = \pi_C(\mathbf{s}_{C,k}; \boldsymbol{\theta})$. $\{a_{C,\mathscr{T},k,m}\}_{m=1}^{\mathcal{C}}$ is learned for each cluster C_m and corresponds to the incentive for increasing the tweet activities of its members, and $\{a_{C,\mathscr{R},k,m,m'}\}_{m,m'=1}^{\mathcal{C}}$ is learned for each *pair* of clusters (m, m') and

indicates how much to incentivize a user of cluster m to retweet the posts of a user in cluster m'.

The value of the network state $\mathbf{s}_{U,k}$ is the total expected reward when in the given state following policies π_C, π_U. Since it is computationally expensive to compute $V(\mathbf{s}_{U,k})$ from future rewards for every possible policy, we approximate the value as a function of the state parameterized by weights $\boldsymbol{\phi}$, i.e. $V(\mathbf{s}_{U,k}) = f(\mathbf{s}_{U,k}; \boldsymbol{\phi})$, (as in [30]). Policy gradient methods are more effective in high dimensional spaces, and can learn continuous policies. Thus, we use advantage actor-critic algorithm (e.g., [28]).

Let there be K stages in the Simulated Training Data (STD) (Fig. 6.1). Given state features at the beginning of stage k (i.e. at time τ_k), we learn policy function π_C to obtain actions to be applied during stage k (i.e. time-interval $[\tau_k, \tau_{k+1})$), using a multi-layer feed-forward neural network. We find intervention actions for the clusters, $\mathbf{a}_{C,\mathscr{T},k}, \mathbf{a}_{C,\mathscr{R},k} = \pi_C(\mathbf{s}_{C,k})$, corresponding to tweet and retweet intensities, respectively, as the output of the neural network (line 6 of Alg. 3). We use two different neural networks, for approximating the policy function π_C, and the value function, $V(\mathbf{s}_{U,k})$. Fig. 6.2 shows the value function approximator for three users, parameterized by weights $\boldsymbol{\phi}$, i.e., $V(\mathbf{s}_{U,k}) = f(\mathbf{s}_{U,k}; \boldsymbol{\phi})$. The input to the value function approximator for stage k, are the state features of users, $\mathbf{s}_{U,k} = [\mathbf{n}_k(F, \mathscr{T}), \mathbf{n}_k(T, \mathscr{T}), \mathbf{n}_k(F, \mathscr{R}), \mathbf{n}_k(T, \mathscr{R}), \mathbf{w}_k]$. The dimensionality of input layer is $5N$, where N is the number of users in the network. We use one hidden layer with $(5N/2)$ nodes to learn the value function. And the output is a scalar value i.e., $V(\mathbf{s}_{U,k})$. We used Adam optimizer with learning rate 0.05 to train this network. Fig. 6.3 shows the policy function approximator for two clusters, with one and two users, respectively, parameterized by weights $\boldsymbol{\theta}$, i.e., $\mathbf{a}_{C,\mathscr{T},k}, \mathbf{a}_{C,\mathscr{R},k} = \pi_C(\mathbf{s}_{C,k}; \boldsymbol{\theta})$. $\mathbf{a}_{C,\mathscr{T},k} = \{a_{C,k,m}\}_{m=1}^{\mathcal{C}}$, and $\mathbf{a}_{G,\mathscr{R},k} = \{a_{C,k,m,m'}\}_{m,m'=1}^{\mathcal{C}}$. The input to the policy function approximator for stage k, are the state features of clusters, $\mathbf{s}_{C,k} = \{\overline{n}_{k,m}(h, z)\}_{m=1}^{\mathcal{C}}$ obtained as the average of the state features of their associated members (users). The dimensionality of input layer is $5\mathcal{C}$, where $\mathcal{C}$ is the number of clusters. We use a hidden layer with $(3\mathcal{C})$ nodes to learn the policy function. And the outputs are a $\mathcal{C}$-dimensional vector

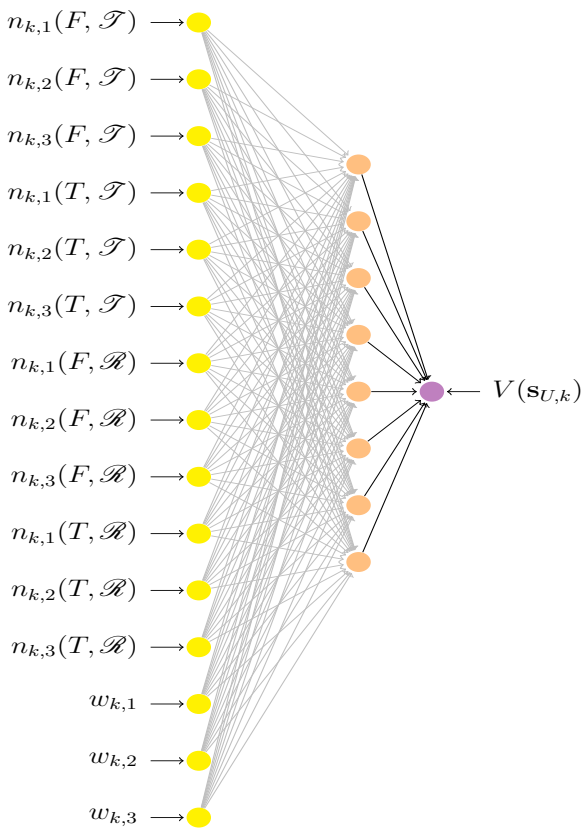


Figure 6.2.: NN for approximating Value Function

corresponding to the interventions for the Tweet MHP for each cluster, and a $\mathcal{C}^2$-dimensional vector corresponding to the interventions for the Retweet MHP for pairs of clusters. We used Adam optimizer with learning rate 0.01 to train this network.

Then, in line 10, we obtain interventions for the users $\mathbf{a}_{U,z,k}$ based on their *variability* from their cluster, using Alg. 4. Alg. 4 computes the interventions for the users by weighting the cluster interventions by their distance to the centroid. Since retweets involve an interaction between pair of users, we consider incentives $\alpha_{U,\mathscr{R},k,i,j}$ for each pair of users, where $\alpha_{U,\mathscr{R},k,i,j}$ is the amount of incentive provided to user i to retweet user j's posts. However, due to high computational cost, we do not want to model an N^2-dimensional MHP for retweet activities. So instead, to reduce it to an $N-$dimensional MHP, we compute the weighted average of incentive actions in line 8. We normalize user actions based on our budget constraint in line 9. Then,

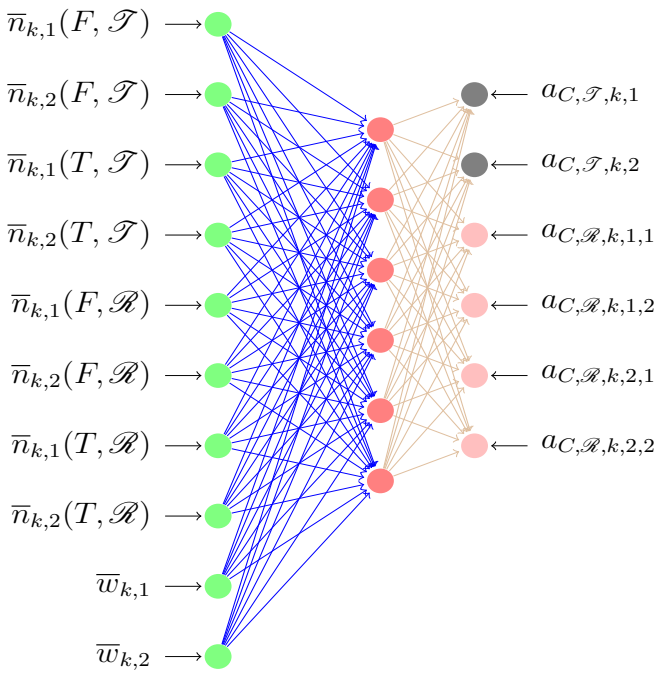


Figure 6.3.: NN for estimating cluster-level actions

Algorithm 4 GetUserInterventions (π_U)

Input: $\mathbf{a}_{C,z,k}, \mathbf{X}_k, \mathbf{Y}_k$

Let $c_{k,i}$ be the cluster to which user i belongs to, at stage k.

/* Incentive per user for Tweet MHP */

$\tilde{a}_{U,\mathscr{T},k,i} = a_{C,\mathscr{T},k,c_{k,i}} ||\mathbf{X}_{k,i} - \mathbf{Y}_{k,c_{k,i}}||_2, \; \forall i \in [1, N]$

/* Incentive, per pair of users for retweets */

$\alpha_{U,\mathscr{R},k,i,j} = a_{C,\mathscr{R},k,c_{k,i},c_{k,j}} ||\mathbf{X}_{k,i} - \mathbf{Y}_{k,c_{k,i}}||_2 ||\mathbf{X}_{k,j} - \mathbf{Y}_{k,c_{k,j}}||_2, \; \forall i, j \in [1, N]$

/* Weighted average to get incentive per user for Retweet MHP */

$\tilde{a}_{U,\mathscr{R},k,i} = \frac{1}{N} \sum_{j=1}^{N} \alpha_{U,\mathscr{R},k,i,j} \Phi_{\mathscr{R},ji}, \; \forall i \in [1, N]$

$\mathbf{a}_{U,z,k} = \frac{\tilde{\mathbf{a}}_{U,z,k}}{||\tilde{\mathbf{a}}_{U,z,k}||_1} \times O_{z,k}$ /* Budget Constraint */

return $\mathbf{a}_{U,z,k}$

we compute the expected reward using these actions, as described in Alg. 5, and the total objective for optimizing policy π_C in Alg. 6.

Algorithm 5 GetExpectedReward

Input: $\mathbf{s}_{U,k}, \mathbf{a}_{U,z,k}, \boldsymbol{\phi}, \gamma$

Compute $\mathbb{E}[R_k(\mathbf{s}_{U,k}, \mathbf{a}_{U,z,k})]$

$\mathbf{s}_{U,k'} = (\mathbb{E}[\mathbf{n}_k(h, \mathscr{T})], \mathbb{E}[\mathbf{n}_k(h, \mathscr{R})], \mathbb{E}[\mathbf{w}_k])$

$V(\mathbf{s}_{U,k'}) = f(\mathbf{s}_{U,k'}; \boldsymbol{\phi})$

$r_k = \mathbb{E}[R_k(\mathbf{s}_{U,k}, \mathbf{a}_{U,\mathscr{T},k}, \mathbf{a}_{U,\mathscr{R},k})] + \gamma V(\mathbf{s}_{U,k'})$

return r_k

Algorithm 6 GetTotalObjective

Input: $\{r_k\}_{k=1}^K, \{\mathbf{s}_{U,k}\}_{k=1}^K, \boldsymbol{\phi}, \gamma$

$L_\theta = 0, \quad L_\phi = 0$

for $k = 1, ..., K$ **do**

    $V(\mathbf{s}_{U,k}) = f(\mathbf{s}_{U,k}; \boldsymbol{\phi})$ /* Value Function */

    Let $D_k = \sum_{j=k}^K \gamma^k r_k$ /* Total Discounted Reward */

    $B_k = D_k - V(\mathbf{s}_{U,k})$ /* Advantage Function */

    $L_\theta = L_\theta + B_k; \quad L_\phi = L_\phi + ||V(\mathbf{s}_{U,k}) - D_k||_2$

end for

$J_\theta = L_\theta, \quad J_\phi = -L_\phi$

return J_θ, J_ϕ

Update Clusters

Using actions $\mathbf{a}_{U,z,k}$ learnt for stage k, we calculate the clustering features (Eq. 6.10-6.11) for the next stage, $\mathbf{X}_{k+1}$. Due to application of the policy, $\mathbf{X}_{k+1,i}$ is different from $\mathbf{X}_{k,i}$. Thus, we need to re-compute the centroids and cluster memberships. Additionally, we want the clusters to be aligned across different stages, so that the policies can be optimized using the neural network, for different clusters across multiple epochs. To achieve this, we define *weighted centroids* that include the effect of the centroids in the previous stage. This helps to ensure that the centroids do not shift much and thus, we can align clusters in stage $k+1$ with those in stage k.

Algorithm 7 UpdateClusters

1: **Input:** $\mathbf{Y}_k, \boldsymbol{\mathcal{M}}_k, \boldsymbol{X}_{k+1}, \delta$
2: Let $\ddot{\boldsymbol{Y}}_{k+1} = \mathbf{Y}_k$, $\boldsymbol{\mathcal{M}}_{k+1} = \boldsymbol{\mathcal{M}}_k$
3: **repeat**
4: Let $\dot{\boldsymbol{y}} = \ddot{\boldsymbol{Y}}_{k+1}$ /* Centroids in previous iteration */
5: $\dot{\boldsymbol{Y}}_{k+1,m} = (\sum_{i=1}^{N} \mathcal{M}_{k+1,i,m}\mathbf{X}_{k+1,i})/(\sum_{i=1}^{N} \mathcal{M}_{k+1,i,m}) \quad \forall m \in [1, \mathcal{C}]$
6: $\ddot{\boldsymbol{Y}}_{k+1,m} = \varepsilon_1 \dot{\boldsymbol{Y}}_{k+1,m} + \varepsilon_2 \mathbf{Y}_{k,m} \quad \forall m \in [1, \mathcal{C}]$
7: $m_i = \arg\max_{m=1}^{\mathcal{C}} ||\mathbf{X}_{k+1,i} - \ddot{\boldsymbol{Y}}_{k+1,m}||^2$
8: $\mathcal{M}_{k+1,i,m_i} = 1$, and $\forall m \neq m_i, \mathcal{M}_{k+1,i,m} = 0$
9: **until** $||\dot{\boldsymbol{y}} - \ddot{\boldsymbol{Y}}_{k+1}||_2 < \delta$ /* Convergence */
10: **return** $\boldsymbol{\mathcal{M}}_{k+1}$

Alg. 7 shows the steps to update clusters. For clustering at stage $k+1$, we begin with the memberships from stage k (line 2). This helps in faster convergence of clusters. Using $\mathbf{X}_{k+1}$, we obtain the updated centroids $\dot{\boldsymbol{Y}}_{k+1,m} \forall m \in [1, \mathcal{C}]$ for stage $k+1$ (line 5). We define $\mathcal{C}$ *weighted centroids* $\ddot{\boldsymbol{Y}}_{k+1,m} = \varepsilon_1 \dot{\boldsymbol{Y}}_{k+1,m} + \varepsilon \mathbf{Y}_{k,m} \forall m \in [1, \mathcal{C}]$, (line 6), $\varepsilon_1 + \varepsilon_2 = 1$. ε_1, ε_2 indicate the importance assigned to the centroids from the previous stage and the current stage, respectively. After updating the centroids, we update the membership matrix and repeat until convergence. Additionally, since the change in policy estimates across epochs reduces as optimization gets closer to convergence, the clustering features (which are dependent on these estimates) do not change much for the same stages across such epochs and we can start to reuse the learned clusters. Similar to simulated annealing, we only update the cluster assignments every $\eta_e \in \mathbb{Z}^+$ epochs gradually increasing η_e as the epoch number increases, which helps speed up convergence of policy learning.

6.4.4 Policy Evaluation

Since the Simulated Evaluation Data (SED) is conditioned on STD, we use the clusters converged at the end of the policy learning to evaluate the learned policy. First, we find intervention actions for the network state obtained from events in SED. Then, we generate events by simulating MHPs after adding interventions to the base intensities for true news diffusion, and use those to compute the following evaluation metric.

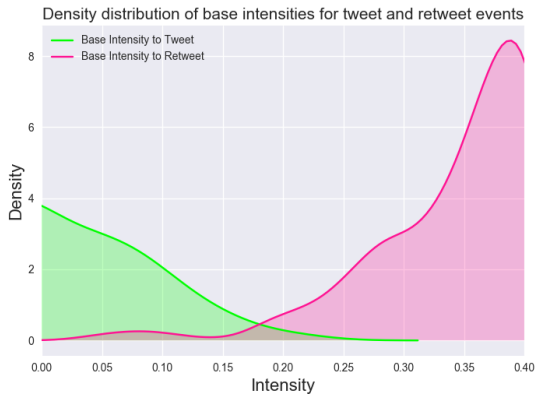
Evaluation Metric

Comparing different methods solely based on reward (e.g., [15]) does not suffice, and a better model is the one that mitigates more distinct number of users [43]. Thus, we multiply the reward by the the fraction of users exposed to fake news that become exposed to true news. This helps to assign more importance to the mitigation of distinct users over mitigation of few users with high exposures. Specifically, performance $\mathcal{P}$ is given as $\sum_{k=1}^{K} R_k \times \frac{|L_{T,k} \cap L_{F,k}|}{|L_{F,k}|}$, where $L_{T,k} = \{i | i \in [1, N], \mathbf{n}_k(T, z) \cdot \mathbf{G}_{.i} > 0\}$ and $L_{F,k} = \{i | i \in [1, N], \mathbf{n}_k(F, z) \cdot \mathbf{G}_{.i} > 0\}$ are the sets of users exposed to true and fake news, respectively, during stage k.

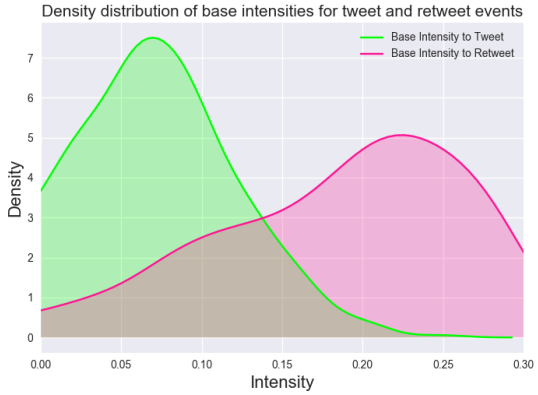
6.5 Experiments

We use real-world datasets, Twitter 2016 and Twitter 2015 [5, 146], with 750 and 2050 users, respectively. We consider time-horizon $T = 40$, divided into 40 stages of $\Delta T = 1$ hour each. The Training Data, STD, SED before and after applying interventions, and Held-Out Data (Fig. 6.1), respectively, correspond to time-intervals, $[0, 10)$, $[10, 20)$, $[20, 30)$, and $[30, 40)$.

First, we test whether news diffusion patterns via tweet and retweet activities are similar in the Twitter data. Figures 6.4 and 6.5 show a comparison of the distribution of respective base intensities across all users. We find that more number of users

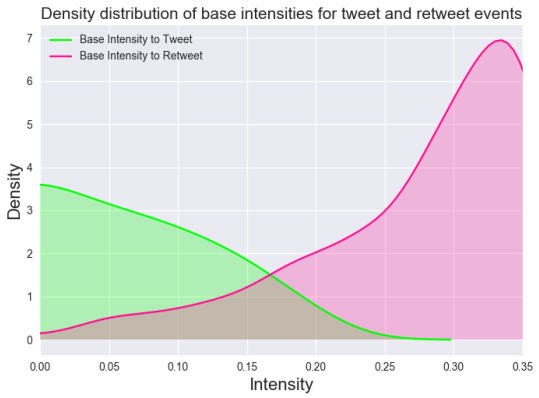


(a) Fake News

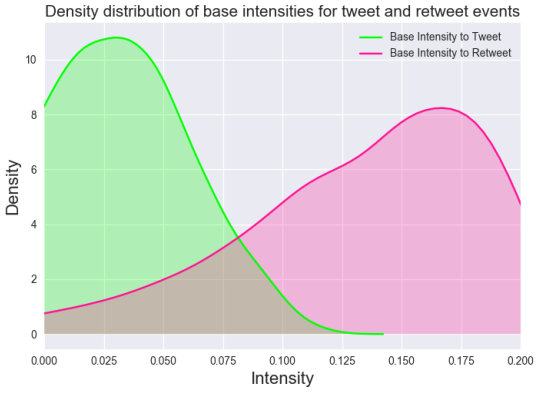


(b) True News

Figure 6.4.: Distribution of Base Intensities for Twitter 2016



(a) Fake News



(b) True News

Figure 6.5.: Distribution of Base Intensities for Twitter 2015

have higher retweet intensities for fake news, compared to true news. Also, the distribution of retweet intensities has more variance compared to tweet intensities. These observations are consistent with [3], and imply that it is important to consider separate processes for tweet and retweet events, as described in Sec. 6.1.

6.5.1 Baselines

We compare our model, which we call **DCPL**, against different baselines described below. We use $O_{z,k} \sim N \cdot \mathcal{U}(0,1)$, and $\gamma = 0.7$. For C-PF, we set $\varepsilon_1 = \varepsilon_2 = 0.5$ to assign equal importance to the centroids from previous and current stages.

Non-Clustering Methods

NC-1

No Clustering. All users in same cluster i.e. $\mathcal{C} = 1$

NC-N

Identical actions for tweet and retweet events. Each user is in separate cluster ($\mathcal{C} = N$) ([43]).

NC-TR

Separate actions learnt for tweet and retweet activities (for true news diffusion), and $\mathcal{C} = N$.

NC-PF

Same as NC-TR, but with clustering features (Eq. 6.10, 6.11) added in the state representation of users, and $\mathcal{C} = N$.

Clustering Based Methods

RND

Randomly assign users to $\mathcal{C}$ static (fixed) groups.

C-NET

Clusters obtained using K-Means++ with network features (degree, closeness centrality, clustering coefficient).

KM-R

Static clusters are obtained using K-Means++ with empirical reward features, and are not updated dynamically.

KM-S

Static clusters as in KM-R, but state features are also used in clustering ([164, 165]).

The input to the policy function approximator (i.e. neural network) requires input and output of fixed dimensions, and hence, we assume fixed number of clusters $\mathcal{C}$. We use the scores of Bayesian Information Criterion, and Within Cluster Sum of Squared Distance to choose $\mathcal{C}$ for each clustering-based method. Table 6.1 reports the number of clusters ($\mathcal{C}$) obtained for different baselines, and we observe $\mathcal{C} \in \{8, 9\}$ for different methods.

Table 6.1.: Number of Clusters ($\mathcal{C}$) for Clustering Based Methods

	DENPL	**KM-R**	**KM-S**	**C-NET**	**RND**
Twitter 2016	9	9	9	8	9
Twitter 2015	8	8	9	8	9

Table 6.2.: Relative Performance (Mean ± Std. Error)

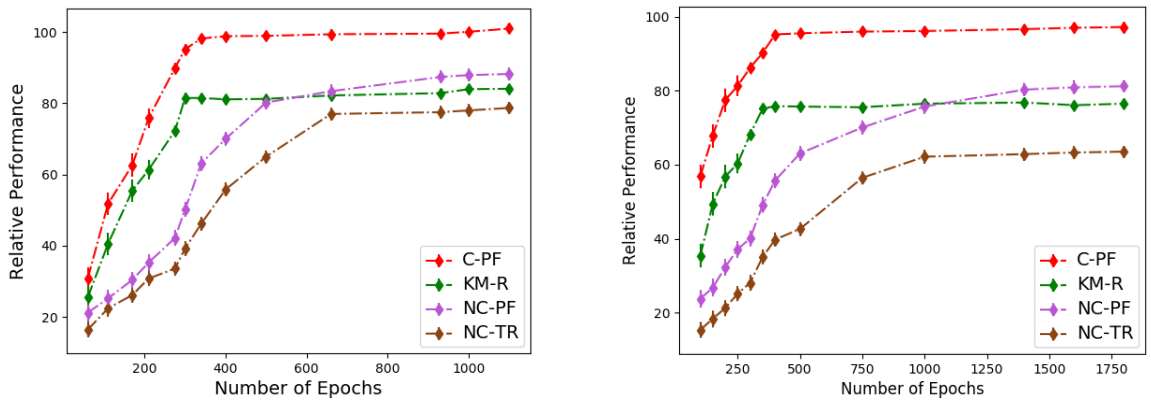
		Twitter 2016	**Twitter 2015**
Clustering	C-PF	**98.19** ± 1.52	**95.175** ± 1.75
Based	KM-R	81.28 ± 1.78	73.98 ± 1.64
Methods	KM-S	78.63 ± 1.82	70.37 ± 1.69
	C-NET	64.07 ± 1.98	51.52 ± 1.72
	RND	55.52 ± 4.23	42.17 ± 4.94
Non-Clustering	NC-PF	87.39 ± 3.46	80.73 ± 3.72
Methods	NC-TR	77.83 ± 3.37	62.76 ± 3.51
	NC-N	67.04 ± 3.21	56.10 ± 3.48
	NC-1	58.68 ± 1.02	47.12 ± 1.10

6.5.2 Results

Table 6.2 shows the relative performance of different methods. Our approach DCPL achieves high correlation between exposures to true and fake news, along with maximizing number of distinct *mitigated* users. Also, NC-TR outperforms NC-N, implying that tweet and retweet events are not identical and it is beneficial to decouple these. The clustering-based methods achieve greater performance than non-clustering methods.

NC-N, NC-TR, NC-PF have larger variance and noisy estimates due to high dimensionality of state/action space. NC-1 has high bias as it doesn't consider differences in user behavior. Clustering based approaches have lower variance, implying that clustering helps to reduce noise in estimates.

Moreover, DCPL that updates cluster assignments dynamically based on policy applied, outperforms those that assume fixed assignments (KM-R, KM-S, C-NET, RND). NC-PF that does not perform clustering also outperforms these, as it adjusts



(a) Twitter 2016 (b) Twitter 2015

Figure 6.6.: Number of Epochs until Convergence

policy based on dynamic user behavior. Thus, we need features indicative of users' *payoff* and *contribution*, apart from state features, to get better estimates.

The performance of KM-R is slightly greater than KM-S, implying that reward based features alone are useful for learning better estimates, without state features. C-NET achieves lower performance than non-clustering baselines, indicating that not all features are useful in clustering users to reflect the amount of incentive needed. The reward based features in DCPL consider network structure implicitly, and are better indicators of users' payoff and contribution.

Fig. 6.6 shows a comparison of the time taken until convergence by different methods. DCPL converges faster than NC-PF and NC-TR, and achieves a greater performance for all epochs. NC-PF outperforms KM-R and NC-TR, but takes longer to converge, implying that the latent features (Eq. 6.10, 6.11) are useful if included in state representation, however, lead to increased computational cost. In DCPL, the information about users' payoff and contribution via clusters, helps in better exploration over the action space without increasing the state space. This helps the model to learn users' effectiveness early on and converge faster to the optimal policy.

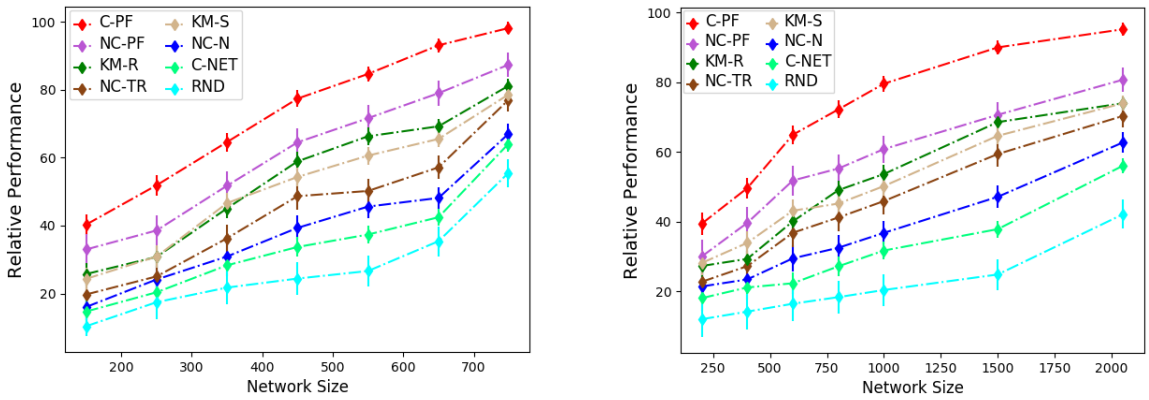
The above results serve as a proof of concept that providing incentives helps to mitigate the impact of fake news. However, since we cannot make real-time interven-

Table 6.3.: Sum of Retweets at $\tau + g$ for Users Selected at τ

MODEL	$\tau^{'} = \tau + 0$		$\tau^{'} = \tau + 2$		$\tau^{'} = \tau + 5$		$\tau^{'} = \tau + 8$	
	S	M	S	M	S	M	S	M
C-PF	**1320.9**	**439.8**	**1289.1**	393.2	**912.5**	**352.4**	**853.3**	**208.2**
KM-R	1181.2	546.6	875.6	381.2	667.4	419.6	603.2	366.8
KM-S	1103.7	510.8	871.2	**311.3**	660.4	415.5	592.2	356.6
C-NET	810.4	691.2	695.3	669.6	517.8	545.7	496.5	453.6
RND	569.9	712.5	485.1	600.6	369.7	706.8	329.2	591.3
NC-PF	1200.4	471.4	890.7	406.2	682.6	397.6	593.2	283.4
NC-TR	1077.2	522.4	862.4	425.8	652.5	450.6	567.4	274.4
NC-N	1003.5	450.8	838.1	378.3	625.7	378.3	551.9	297.6

tions, we also compare different methods by measuring the impact of nodes selected for intervention, in terms of the people they actually reached in the Held-Out Data, as in [43]. Let $S(\tau)$ refer to the the set of users *selected* to spread true news by time τ, according to the model, i.e., $S(\tau) = \{i|(\mathcal{N}_i(\tau, T, z) - \mathcal{N}_i(2K, T, z)) > 0\}$ $\tau \in [20, 30)$, and the remaining users are considered *missed* ($M(\tau)$) by the model. We calculate the total number of users who retweeted the posts of users in $S(\tau)$ and $M(\tau)$ between time $[\tau^{'}, \tau^{'} + \Delta)$ where $\tau^{'} = \tau + g$, and $g = \{0, 2, 5, 8\}$ indicates the gap or number of stages after which we want to measure the impact (in the future). We considered different values of $\Delta \in \{1, 2, 3, 4, 5\}$ and Table 6.3 reports the average. We see that the impact of selected nodes (S) is greater than that of missed nodes (M) for DCPL by a large margin.

We conducted additional experiments to explore the effects of network characteristics on performance. We down-sampled the datasets to compare the performance of different approaches as a function of network size (number of users). Fig. 6.7 shows



(a) Twitter 2016

(b) Twitter 2015

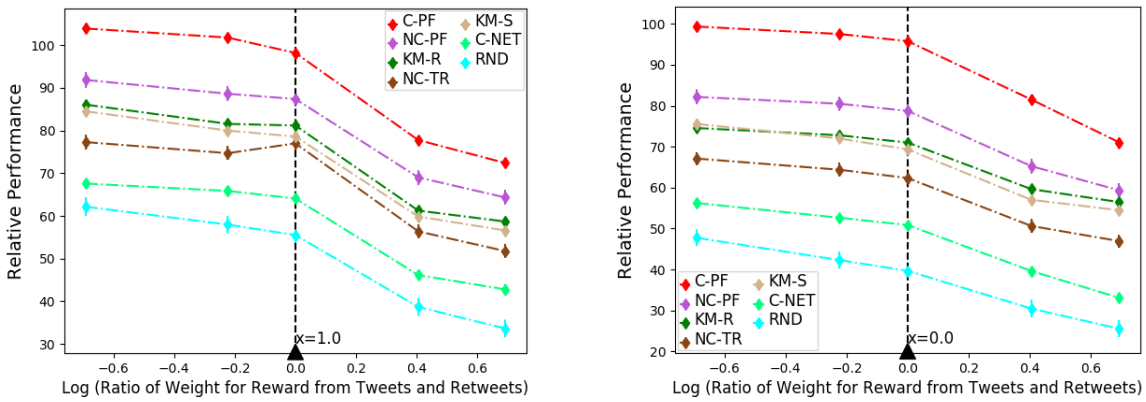
Figure 6.7.: Relative Performance vs Network Size

that the performance of all methods decreases with a decrease in network size, and our method DCPL outperforms for all network sizes considered.

We vary ζ, the relative importance of reward from tweet and retweet activities. Fig. 6.8 shows that with an increase in this ratio, that is a decrease in the contribution by retweet events, the performance decreases for all methods. We also observe a relatively steep decrease in the performance after a ratio of 1 that corresponds to equal importance of tweet and retweet events. This suggests that retweets contribute more to the total reward than tweets, which is also consistent with [3].

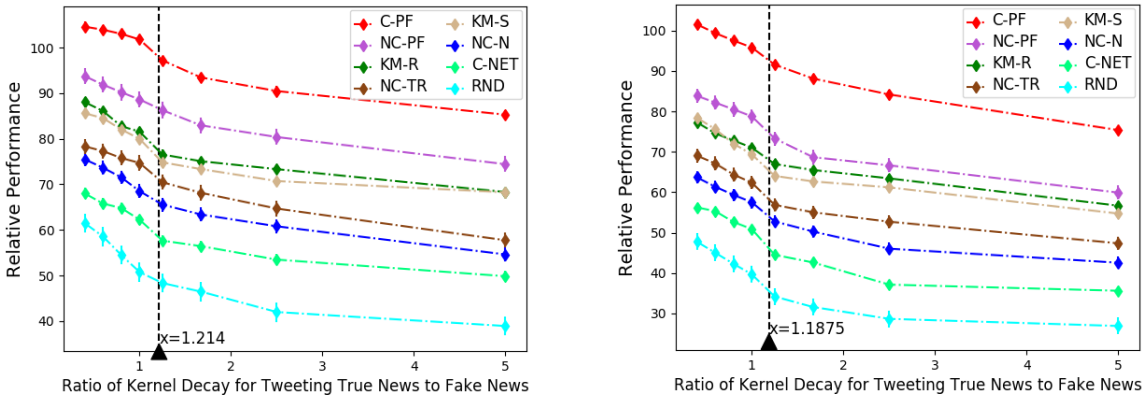
Figures 6.9 and 6.10, respectively, show the change in the performance with respect to change in the kernel decay parameter, for tweeting and retweeting fake and true news. We keep the decay parameter for Tweet MHP fixed, while varying the decay parameter for Retweet MHP, and vice-versa. As this ratio increases, the performance decreases, for both tweet and retweet activities, due to presence of fewer true news events than fake, in later stages. We also observe a slightly more steep decrease in performance for retweet activities, compared to tweet activities, implying that retweets have a major contribution in the reward, compared to tweets.

We also conducted additional experiments to explore characteristics of the learned clusters. Fig. 6.11 shows the relative performance of different clustering-based meth-



(a) Twitter 2016 (b) Twitter 2015

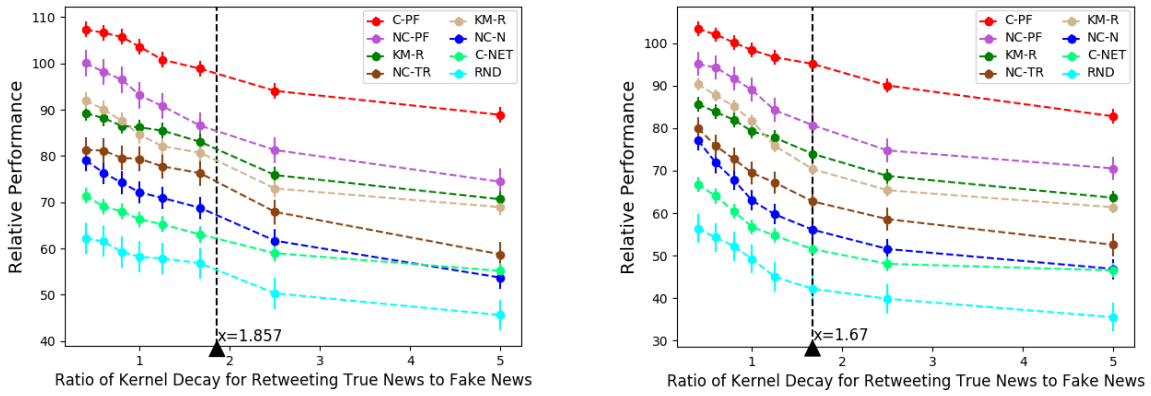
Figure 6.8.: Relative Importance of Tweets and Retweets



(a) Twitter 2016 (b) Twitter 2015

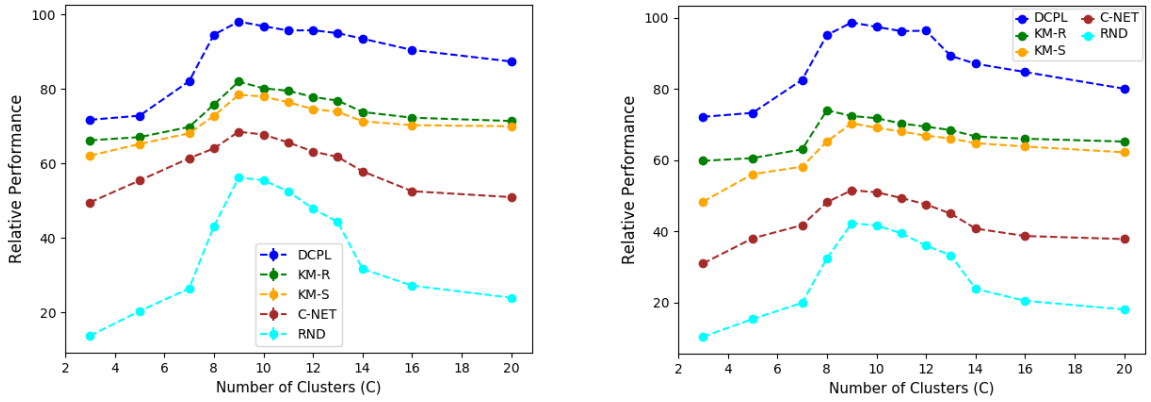
Figure 6.9.: Ratio of decay for tweeting true and fake news

ods with respect to the number of clusters $\mathcal{C}$. We observe that the methods achieve greater performance for number of clusters 8 and 9, which confirms the selection of $\mathcal{C}$ based on BIC and WC-SSD scores (Sec. 6.5.1). We observe that the methods perform better for smaller number of clusters in the range $\mathcal{C} \in [8, 12]$. Moreover, DCPL outperforms other baselines for all values of $\mathcal{C}$, which is due to the fact that it captures dynamic user behavior by re-assigning users to clusters at different stages.



(a) Twitter 2016 (b) Twitter 2015

Figure 6.10.: Ratio of decay for retweeting true and fake news



(a) Twitter 2016 (b) Twitter 2015

Figure 6.11.: Relative Performance vs Number of Clusters

We evaluate cluster alignment across different stages, in our method DCPL by comparing the clusters at stage k with those obtained in the previous stage $k-1$. Fig. 6.12 shows the Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) scores. For stage 1, we compare with the initial clusters obtained using the reward computed from training data. We see that both NMI and ARI scores are high, and this presents a proof of concept that clusters of DCPL are indeed aligned.

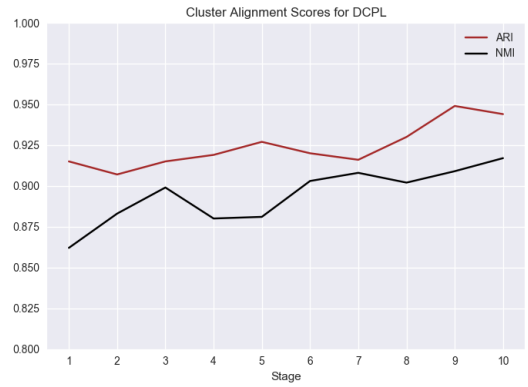
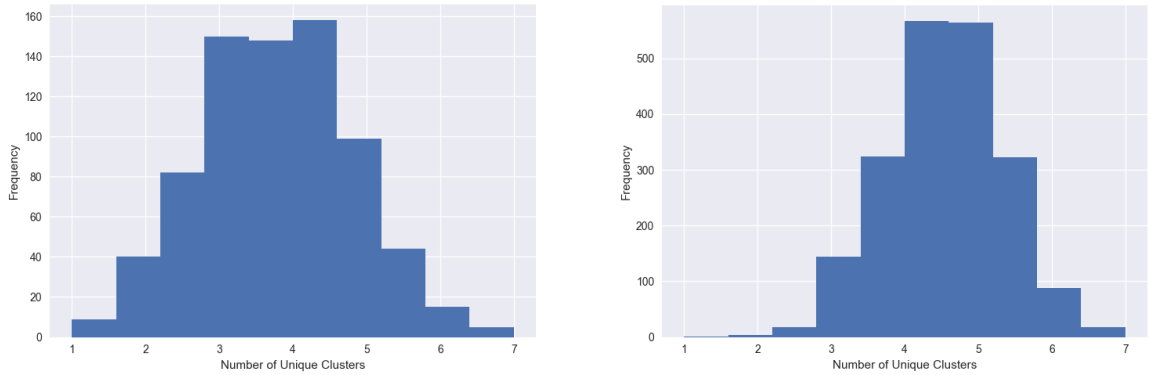


Figure 6.12.: Cluster Alignment Scores for DCPL

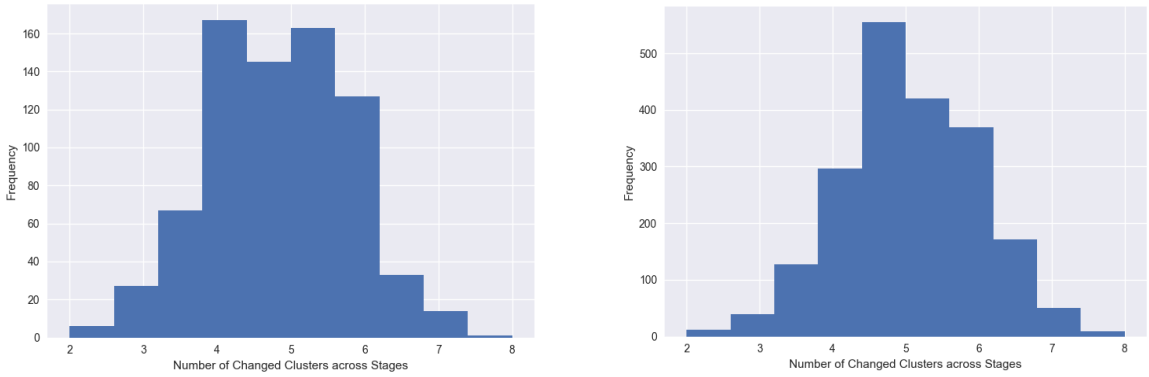


(a) Twitter 2016

(b) Twitter 2015

Figure 6.13.: Number of Unique Clusters across Users

Additionally, we also analyze the movement of users between different clusters across multiple stages, in DCPL. Figures 6.13 and 6.14, respectively, show histograms for the number of unique clusters that a user is in, and the number of times a user changes clusters, across all the stages in the learning phase. We see that majority of the users change clusters 4-6 times. This justifies our conjecture that there is a change in user behavior (features) due to the policy applied in the past, and it is important to capture this by updating the cluster assignments dynamically. We also conclude that users do not return back to the same clusters that they were in the previous stages.

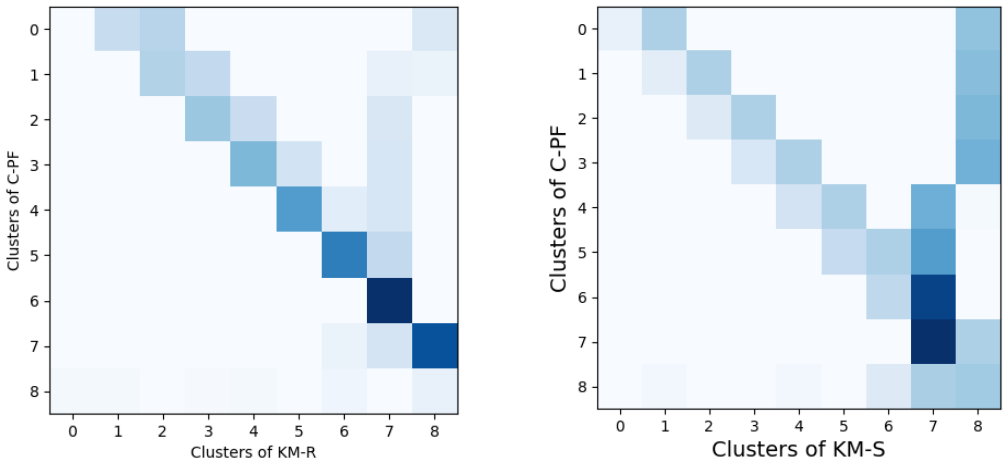


(a) Twitter 2016

(b) Twitter 2015

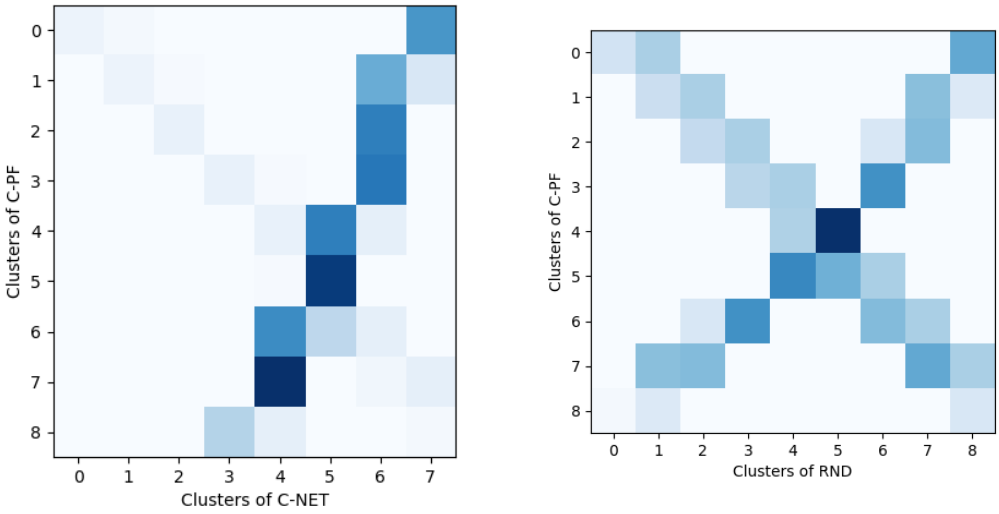
Figure 6.14.: Number of Changed Clusters per User across Stages

Fig. 6.15 shows contingency matrices comparing the clusters obtained by our method DCPL with those obtained by other baselines. Specifically, the rows of the contingency matrix are ordered by the clusters in DCPL, and the columns are ordered by the clusters in the baseline method. The contingency matrix reports the intersection cardinality (number of common users) for every cluster pair from the two methods. We find that clusters of DCPL are most similar to those of KM-R, followed by KM-S, C-NET, RND, consistent with the decreasing order of these methods, in terms of maximizing reward.



(a) KM-R

(b) KM-S



(c) C-NET

(d) RND

Figure 6.15.: Contingency Matrix (C-PF vs other baselines)

7 TOWARDS DECENTRALIZED SOCIAL REINFORCEMENT LEARNING VIA EGO-NETWORK EXTRAPOLATION

7.1 Introduction

We consider a multi-agent reinforcement learning scenario in social networks, where the agents are users trying to optimize long term reward for their actions. The key challenge in this scenario is that individual policies need to account for dependencies throughout the network, as users' actions and reactions are influenced by the activities of their neighbors. For example, in Twitter if one user tweets more about a certain topic, that may influence their Followers to tweet more on the same topic. Thus, user interactions lead to *network dependencies* between activities (due to peer-influence).

Moreover, many online social network relations are directed and as such the network information may only flow in one-direction [167]. For example, Twitter links correspond to Followee-Follower relationships. A Follower can observe the activities of her Followees, however, the information does not flow in the opposite direction, unless the Followee also follows the Follower. Due to the directed nature of these interactions, the environment is partially observable to a user (agent), and we refer to the user's local network as a *partially observable ego-network*.

In these social network settings, there are a number of decision making applications that can be formulated as agents optimizing *individual rewards*, which depend on activities in the larger network context. For example, a user in a social network may want choose when and what to post to maximize her influence or visibility among her followers (e.g., [44]), or an agent receives individual credit based on her contribution in credit assignment and resource allocation tasks (e.g., [35]). The perception of users about the popularity of a certain attribute (e.g., news topic) is governed by the

perception of her local neighbors, and is much different than global prevalence of the attribute in the complete network [167, 168]. Thus, the individual reward of a user might be different from that of other users in the network, depending on her peers and local network structure. To formalize this, we consider a setting where each user receives a separate local reward that depends on her activities and the activities of related users in her local neighborhood.

Recent work on multi-agent reinforcement learning (MARL) for social network settings [15, 43, 169] has assumed a fully observable environment where each agent can observe the activities of all other agents in the network, with a common shared reward between all agents to be optimized. They employ a joint model (centralized training) to capture inter-agent dependencies by learning the collective action of all users conditioned on the complete network state across all users (centralized execution). Centralized learning also allows to share samples and policy parameters between agents, and thus, overcomes the problem of insufficient samples for each user [102], leading to lower variance in estimates.

However, in our setting each user receives a different reward and has a different partial observation of the environment—thus we cannot employ previous approaches based on centralized learning and centralized execution. In addition, the joint state/action space grows larger with the number of agents, and thus, centralized learning with thousands of agents is computationally intensive [170]. In contrast, decentralized learning and decentralized execution focuses on learning a separate policy for every agent individually, and actions are obtained conditioned solely on the local state and observation of the agent. Learning the policies independently allows to preserve the privacy, i.e., limit the data sharing between users as agents do not need to share their local state, observation or reward with other users in the network [89]. However, decentralized training does not scale for large number of agents, as it is impractical to learn thousands of complex policy functions [169]. Additionally, due to sparse interaction data in social networks, the number of samples available for each user in decentralized learning is quite sparse, which would result in large errors due

to variance. Specifically, with decentralized learning/execution it will be challenging to learn accurate policies as each agent does not have sufficient information available to capture inter-agent dependencies.

To address these challenges, in this work we propose to perform *partially centralized learning* with decentralized execution. Specifically, we learn conditioned only on the local state and observation of each user in order to apply the learned policies in decentralized execution. For learning, we consider a single policy function that maps the local state of an agent to her actions. We use parameter sharing to learn this function across users, which offsets the data sparsity issues that would arise in fully decentralized learning. By only sharing the model parameters sequentially, and not the trajectories (state/action/reward) of each user, we aim to provide more autonomy to agents and safeguard their privacy (limit the data shared between agents). We consider a common neural network and agents access the network in a sequence. At a given iteration, only a single agent learns and updates the shared parameters based only on her state, observation and reward.

Given the partial observations of each user, to learn accurate policies, we propose to utilize *ego-network extrapolation* to improvise the estimates of the hidden state information, and exploit the local network structure, relations, and interactions to learn inter-agent dependencies. In a social network with Followee-Follower relationship, each user has two roles—a Followee to certain users, and a Follower of certain users (as shown in Fig. 7.1). We utilize this observation to locally learn the dependency between Followees and Followers. A user can only observe the state of her Followees, and not those of her Followers or other users in the network. However, the reward that a user obtains depends on the activities of her Followers' other Followees as well whom she cannot observe. Now, the activities of a Follower depends on the activities of her Followees. Thus, our idea is that by estimating the activities (i.e., state) of her Followers, a user can approximate the impact of the activities of her Followers' other Followees (whom she cannot observe). Therefore, we propose to estimate the Followers' states, in order to improve the policy for a user. Our

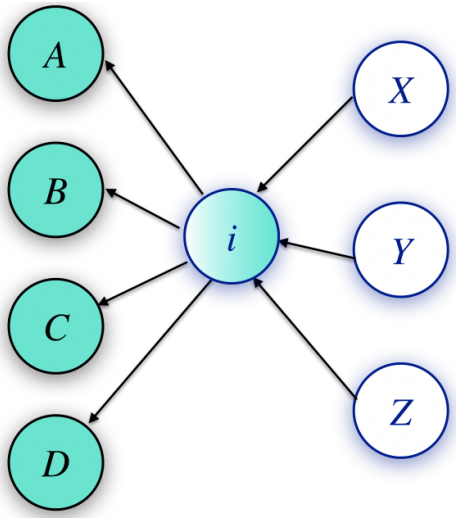


Figure 7.1.: Partially Observable Ego-network of a user i. Teal color corresponds to Followees, and White color corresponds to Followers of a user. User i is a Follower to her Followees A, B, C, D whose activities she can observe, and a Followee of her Followers X, Y, Z whose activities she cannot observe.

key insight is to perform ego-network extrapolation over the local neighborhood of a user, by first learning the dependency from the activities of her Followees, and then extrapolating those to estimate the activities of her Followers. To the best of our knowledge, this is the first MARL approach to exploit the relations between users in a *partially observable* social network, by transferring the knowledge learnt from one set of users to another set of users, and estimate the hidden environment information for policy learning. Note that the mapping between a user's Followees and the user (as a Follower) is learnt locally by the user, and is a many-to-one mapping. Thus, the challenge is to extrapolate/project this mapping from the user (as a Followee) to her Followers, which is one-to-many projection. To overcome this challenge, we capture the reciprocity in user interactions, to learn a many-to-many mapping over the set of users, and that can be easily projected to a many-to-many mapping, leading to better estimates. Additionally, different Followees have different impact on the activities of their Followers, and this impact changes over time based on the dynamic user

activities and interactions. We incorporate the dynamic peer-influence by learning the *attention* between activities of each Followee-Follower pair, to further improvise the estimates of the hidden state of the environment, and thus, the policy for each user.

We refer to our approach as Decentralized Ego-Network Policy Learning (DENPL) and evaluate performance compared to different centralized and decentralized learning approaches, using two real-world Twitter datasets. Compared to other baselines, our approach DENPL achieves performance equivalent to that of the centralized learning method and other approaches that assume full observability of the complete network state. Experiments show that sequential update of parameters by each user individually improves the sample efficiency per user, resulting in more accurate policy estimates.

7.2 Related Work

[43] developed centralized learning based solutions for Social RL problems that use the collective state, actions and feedback of all agents, to capture inter-agent dependencies. However, this is computationally intensive for large networks since the joint state/action space grows with the number of agents. [169] proposed a dynamic clustering-based approach to reduce the effective number of policies that overcomes the problem of high dimensional spaces and sparse interactions. However, centralized learning is applicable only in fully observable environments, where the complete network state is available to all agents. Moreover, these consider a common reward for all agents, and centralized learning and centralized execution (e.g., [71]) is equivalent to single agent RL, and it is easier and faster to learn accurate policy estimates given the complete network state.

While traditional MARL approaches do not scale for large numbers of agents in social networks (see e.g., [169]), there is previous work on MARL scenarios that optimize individual rewards in a larger, dependent context. Cooperative MARL tasks

consider a common reward for all users (e.g., [89, 102, 118]). In our problem however, each user receives a separate local reward that depends on her actions and the actions of other users whom she cannot observe. Additionally, our problem is different from competitive MARL (e.g., [171]), as we do not consider conflicting goals or a fixed shared resource across all users. To learn actions based on local reward, some previous work used fully decentralized learning (e.g., [172]). However, this does not scale to our setting since training a complex model for each user independently is impractical for thousands of agents.

Some other decentralized learning approaches (e.g., [89,118]) assume a global state shared across users and learns actions conditioned on the global state. In most of the MARL approaches for partially observable domains that considers both local state and reward, the hidden state of the environment becomes available to an agent within a short period of time, i.e., before the time-horizon of the task. Thus, these approaches can incorporate relevant state information as history for policy learning (e.g., [58, 59]). Additionally, these consider small number of agents, and hence the memory requirements for storing the history for all agents is lower. In contrast, due to the directed nature of user interactions in our problem setting, an agent can only directly observe Followees, and can only observe Followers if there is a directed path linking them through other agents. This means that even if state information was passed via neighbors in the network, the complete network state would likely not be available before the finite time-horizon. And even if it could, storing complete network trajectory information for a large number of users would be space-prohibitive. Thus, the relevant state information cannot be utilized by the user as history.

Recent work has considered agents connected in a graph with local rewards [57]. However their approach is restricted to discrete action spaces and dense undirected graphs. [89, 118] assume a shared reward between all agents, and develop a decentralized approach based on local message passing to estimate the global reward using estimates from direct neighbors. However, they assume strongly connected graphs for small number of agents where message passing can converge. [102] considered central-

ized learning and decentralized execution through the use of a centralized controller, and parameter sharing between agents for policy learning. However, the shared controller has access to the entire history of state, action and reward for each user. Thus, this does not safeguard the privacy of users in real-world social networks.

7.3 Problem Definition

We consider a social network setting with N users, where each user $i \in \{1, ..., N\}$ is an agent. Users have a Followee-Follower relationship, and interact via d different network activities (e.g., tweet, retweet, like). We represent the followers adjacency matrix for the social network graph using $\mathbf{G}$, where $G_{ji} = 1$ if i follows j, and 0 otherwise. We consider a directed social network graph and thus, the direction of information flow is only in one direction. Due to this, an agent can observe the activities of her Followees $\mathbf{G}_{.i}$, but she can't observe the activities of her Followers $\mathbf{G}_{i.}$. Thus, the environment is partially observable to an agent, and we refer to this as a user's partially observable ego-network.

Let the *state* of a user i, $\mathbf{s}_i \in \mathbb{R}^d$ corresponds to d network activities that she performs, and her local *observation* $\mathbf{o}_i$ is the partial view of the environment that corresponds to the activities of her Followees (described in more detail in Section 7.4.3). Given the individual state and observation, each user learns a policy $\pi_i : \mathbf{s}_i \rightarrow \mathbf{a}_i$ to obtain *actions* that maximize her *local reward* R_i (received from external system). A quantitative measure of the impact of fake and true news is the number of people exposed. We define the reward received by a user as the correlation between exposures to fake and true news among her Followers, based on the idea that users exposed more to true news must be exposed more to fake news [15], along with the number of distinct *mitigated* Followers. In addition, there is a penalty P_i for each user that corresponds to the cost or some amount of effort that a user needs to spend to post.

Our data contains a temporal stream of events with the time horizon $[0,\ \mathcal{T})$ divided into K stages, each of time-interval ΔT, where stage $k \in [1, K]$ corresponds to the time-interval $[\tau_k, \tau_{k+1})$. Users interact via three types of network activities, i.e., tweets ($\mathscr{T}$), retweets ($\mathscr{R}$), and likes ($\mathscr{L}$). Each activity corresponds to an event. The tweet or retweet events are represented using $e = (t, i, h, z)$ where t is the time-stamp at which user i shares a post of type $z = \mathscr{T}$ or $\mathscr{R}$, with label $h = F$ (Fake) or T (True). We represent the Like events as $l(u, i, t)$ indicating user i likes user u's post at time t. We assume that users know the type of news posted by their Followees. Let $\mathcal{N}_i(t, h, z)$ represent the number of times user i shares news of type z = $\mathscr{T}$ or $\mathscr{R}$, with label $h = F$ or T, up to time t. Then, the number of times a user i is exposed to news up to time t corresponds to $\sum_{z=\{\mathscr{T},\mathscr{R}\}} \sum_{j \in \mathbf{G}_{\cdot i}} \mathcal{N}_j(t, h, z)$. Let $W_{j,i}(t)$ be the number of likes provided by user i to user j upto time t. The goal of each user is to mitigate the impact of fake news among her Followers, that is realized algorithmically by increasing their respective sharing rate for true news diffusion in a targeted fashion—such that her Followers receive as much true news as fake (i.e., $\sum_{z=\{\mathscr{T},\mathscr{R}\}} \sum_{j \in \mathbf{G}_{\cdot i}} \mathcal{N}_j(t, F, z) \simeq \sum_{z=\{\mathscr{T},\mathscr{R}\}} \sum_{j \in \mathbf{G}_{\cdot i}} \mathcal{N}_j(t, T, z)$).

7.4 Approach

7.4.1 Overview

Fig. 7.2 illustrates the different components of our system. We learn a set of Multivariate Hawkes Processes (MHP) for different network activities, from the training data. We use the MHP models to simulate additional training data, i.e. events, for learning the policy. We map the excitation events to states and observations in a Partially Observable Markov Decision Process (POMDP), where each user learns the amount by which she needs to increment her *intensity function* for true news diffusion. Our policy learning algorithm is described in detail in Alg. 8. For learning the policy for a user i, our idea is to learn the dependency between the activities of Followees and Followers (line 3), and utilize it to estimate the hidden state of i's Followers, using the

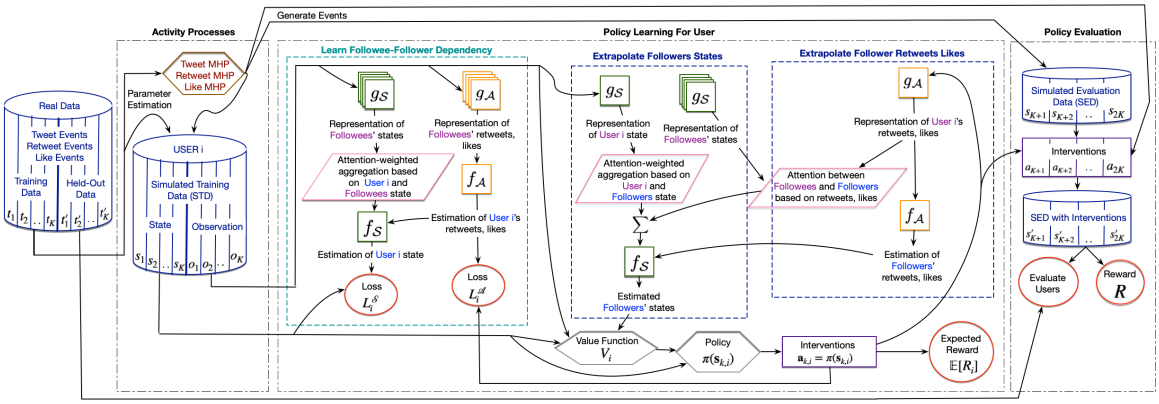


Figure 7.2.: Overview of policy learning for a user

state of i's Followees. Specifically, we first learn a model $g_{\mathcal{S}}$ to obtain a generalized representation for the state of the Followees. Then, we learn another model $f_{\mathcal{S}}$ that learns to estimate the state of user i, $\mathbf{s}_{k,i}$ from an attention-weighted aggregate of her Followees' generalized state representations. We learn a many-to-one mapping from the state of Followees to a user's state, however, while extrapolating the states of Followers from the user's state, it is a one-to-many mapping. This results in less accurate estimates of Followers' states, since there is only one input signal to estimate the states of multiple Followers.

To address this, we utilize the pairwise user interactions, i.e., the retweets and likes that a user provides to her Followees, to learn a many-to-many mapping that can be used to effectively extrapolate the Followers' states. We learn a model $g_{\mathcal{A}}$ to obtain a generalized representation for the retweet and likes provided by user i's Followees (to their Followees), and another model $f_{\mathcal{A}}$ that learns to estimate the retweets and likes provided by user i to her Followees, from her Followees' generalized representation for retweets and likes. After learning these models using the activities of user i and her Followees, we use these to extrapolate the states of user i's Followers from user i's state and observation (lines 5,7). Then, the state of user i, along with her Followees' states, and her Followers' estimated states obtained above, is used to approximate the value of user i's activities V_i (line 9) and compute the expected

reward r_i (line 17), to learn policy π. To evaluate the estimated policy, we simulate data again from the MHPs. Using the learned policy $\hat{\pi}$, we obtain actions that are added to the MHP intensity functions to generate evaluation data, which is then

Algorithm 8 PolicyLearningForUser($i, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_V, \boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im$)

1: /* User i obtains her actions based on her state, using policy π */
2: $\mathbf{a}_{z,k,i} = \pi(\mathbf{s}_{k,i}; \boldsymbol{\theta}_\pi)$ /* Decentralized Execution */
3: $\boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im$ = LearnFolloweeFollowerDependency($i, \mathbf{s}_{k,i}, \mathbf{o}_{k,i}, \mathbf{a}_{z,k,i}, \{\mathbf{a}_{z,k-1,j}\}_{j \in \mathbf{G}_{.i}}, \boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im$) (Alg. 11)
4: /* User estimates her Followers' retweets/likes for stage k */
5: $\{(\hat{n}_{\mathscr{R},k,i,m}, \hat{w}_{k,i,m}\}_{m \in \mathbf{G}_{i.}}, \{\alpha^\mathcal{A}_{k,j,m})\}_{m \in \mathbf{G}_{i.}, j \in \mathbf{G}_{.i}}$ = ExtrapolateFollowersRetweetsLikes($i, \mathbf{a}_{z,k-1,i}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{A}$) (Alg. 13)
6: /* User estimates her Followers' states for stage k */
7: $\{\hat{\mathbf{s}}_{k,m}\}_{m \in \mathbf{G}_{i.}}$ = ExtrapolateFollowersStates($i, \mathbf{s}_{k,i}, \{\mathbf{s}_{k,j}\}_{j \in \mathbf{G}_{.i}}, \{(\hat{n}_{\mathscr{R},k,i,m}, \hat{w}_{k,i,m}\}_{m \in \mathbf{G}_{i.}}, \{\alpha^\mathcal{A}_{k,j,m})\}_{m \in \mathbf{G}_{i.}, j \in \mathbf{G}_{.i}}, \boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\Im$) (Alg. 12)
8: /* User estimates her Value Function */
9: $V_{k,i} = f_V(\mathbf{s}_{k,i}, \{\mathbf{s}_{k,j}\}_{j \in \mathbf{G}_{.i}}, \{\hat{\mathbf{s}}_{k,m}\}_{m \in \mathbf{G}_{i.}}; \boldsymbol{\theta}_V)$
10: /* User estimates her expected state and observation for next stage */
11: $\mathbf{s}_{k',i} = (\mathbb{E}[n_{k,i}(h,z)])$, $\{\mathbf{s}_{k',j}\}_{j \in \mathbf{G}_{.i}} = \{(\mathbb{E}[n_{k,j}(h,z)])\}_{j \in \mathbf{G}_{.i}}$
12: /* User estimates her Followers' expected states for next stage */
13: $\{(\hat{n}_{\mathscr{R},k',i,m}, \hat{w}_{k',i,m}\}_{m \in \mathbf{G}_{i.}}, \{\alpha^\mathcal{A}_{k',j,m})\}_{m \in \mathbf{G}_{i.}, j \in \mathbf{G}_{.i}}$ = ExtrapolateFollowerRetweetsLikes($i, \mathbf{a}_{z,k,i}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{A}$) (Alg. 13)
14: $\{\hat{\mathbf{s}}_{k',m}\}_{m \in \mathbf{G}_{i.}}$ = ExtrapolateFollowersStates($i, \mathbf{s}_{k',i}, \{\mathbf{s}_{k',j}\}_{j \in \mathbf{G}_{.i}}, \{(\hat{n}_{\mathscr{R},k',i,m}, \hat{w}_{k',i,m}\}_{m \in \mathbf{G}_{i.}}, \{\alpha^\mathcal{A}_{k',j,m})\}_{m \in \mathbf{G}_{i.}, j \in \mathbf{G}_{.i}}, \boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\Im$)
15: /* Compute expected reward */
16: $V_{k',i} = f_V(\mathbf{s}_{k',i}, \{\mathbf{s}_{k',j}\}_{j \in \mathbf{G}_{.i}}, \{\hat{\mathbf{s}}_{k',m}\}_{m \in \mathbf{G}_{i.}}; \boldsymbol{\theta}_V)$
17: $r_{k,i} = \mathbb{E}[R_{k,i}] + \gamma V_{k',i}$
18: Store $r_{k,i}, V_{k,i}$
19: **return** $\boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im$

used to measure empirical reward. We also assess the effectiveness of the users who choose to increment their intensities to promote true news, under the policy learned by our model, by measuring the number of retweets accumulated by them, in held out training data. We describe each component next.

7.4.2 Activity Processes

We use N-dimensional MHPs ([37]) to model user activities and simulate environment dynamics. Let $\lambda_{h,\mathscr{T},i}$ be the intensity function governing the tweet events of user i:

$$\lambda_{h,\mathscr{T},i}(t) = \mu_{h,\mathscr{T},i} + \sum\nolimits_{j=1}^{N} \int_0^t \Phi_{ji} \ (\omega_{h,\mathscr{T}} e^{-\omega_{h,\mathscr{T}} t}) \ d\mathcal{N}_j(s, h, \mathscr{T})$$

where, where $h = F$ or T, and the integral is over time, and s is used as placeholder for limits {0,t}. $\mu_{h,\mathscr{T},i}$ is the base exogenous intensity of user i to tweet, Φ_{ji} is a kernel adjacency matrix estimated from the training data, and $\omega_{h,\mathscr{T}} e^{-\omega_{h,\mathscr{T}} t}$ is exponential Hawkes kernel. To obtain $\mathcal{N}_i(t, h, \mathscr{T})$, and $W_{j,i}(t)$, respectively, we use the *Tweet MHP* and *Like MHP* models proposed in Chapter 5. Let $a_{\mathscr{T},k,i}$ be a constant intervention action for user i at stage k (time $t \in [\tau_k, \tau_{k+1})$), added to the her base intensity to tweet, for true news diffusion (i.e. $h = T$).

$$\lambda_{T,\mathscr{T},i}(t) = \mu_{h,\mathscr{T},i} + a_{\mathscr{T},k,i} + \sum\nolimits_{j=1}^{N} \int_0^t \Phi_{ji} \ (\omega_{h,\mathscr{T}} e^{-\omega_{h,\mathscr{T}} t}) \ d\mathcal{N}_j(s, h, \mathscr{T})$$

To capture the reciprocity in user interactions, we define *Retweet MHP* as follows.

$$\lambda_{h,\mathscr{R}v,i}(t) = \mu_{h,\mathscr{R},i} + \sum\nolimits_{j \in \mathbf{G}_{\cdot i}}^{N} \int_0^t G_{ji} \ (\omega_{\mathscr{R}} e^{-\omega_{\mathscr{R}} t}) \ d\mathcal{N}_{vj}(s, h, \mathscr{R})$$

where, $\lambda_{h,\mathscr{R},k+1,v,i}(t)$ is the intensity function for user i to retweet the post of her Followee v, and $\mathcal{N}_{vj}(t, h, \mathscr{R})$ is the number of times user j retweets the posts of user v up to time t. The number of retweets of a user j (Sec. 7.3) can be obtained as $\mathcal{N}_j(t, h, \mathscr{R}) = \sum_{v \in \mathbf{G}_{\cdot j}} \mathcal{N}_{vj}(t, h, \mathscr{R})$. Similar to [154], we aggregate the retweets by

Followees of user i to the post of user v, based on the idea that the more frequently the Followees of i retweet v's posts, the more she tends to retweet v's posts. When i retweets v's post, $\mathcal{N}_{vi}(t, h, \mathscr{R})$ gets incremented, that furthur increases the likelihood of retweeting v's posts among the Followers of i. Let $a_{\mathscr{R},k,v,i}$ be a constant intervention action for user i to retweet Followee v's posts, at stage k, added to the her base intensity, for true news diffusion.

$$\lambda_{h,\mathscr{R},v,i}(t) = \mu_{h,\mathscr{R},i} + a_{\mathscr{R},k,v,i} + \sum\nolimits_{j \in \mathbf{G}_{\cdot i}}^{N} \int_0^t G_{ji}\ (\omega_{\mathscr{R}} e^{-\omega_{\mathscr{R}} t})\ d\mathcal{N}_{vj}(s)$$

To simulate the event data, we interleave the MHPs to generate tweet events, then retweet events, and then like events.

7.4.3 Formulation Details

State Let $n_{k,i}(h, z) = \mathcal{N}_i(\tau_k, h, z) – \mathcal{N}_i(\tau_{k-1}, h, z)$ represent the number of times user i shares news in stage $k-1$. Let $\mathbf{s}_{k,i}$ be the state of agent i, that comprises of the number of tweets, and retweets that user i has made corresponding to fake and true news in stage k, i.e., $\mathbf{s}_{k,i} = (n_{k,i}(T, \mathscr{T}), n_{k,i}(F, \mathscr{T}), n_{k,i}(T, \mathscr{R}), n_{k,i}(F, \mathscr{R}))$.

Observation Each user can observe the state of her Followees, along with the number of retweets and likes that her Followees have provided to their Followees. Let $w_{k,j,i}$ be the number of likes provided by user i to her Followee j in stage k, that can be computed as $w_{k,j,i} = W_{j,i}(\tau_{k+1}) - W_{j,i}(\tau_k)$, where we obtain $W_{j,i}(t)$ using *Like MHP*. Let $n_{\mathscr{R},k,j,i}$ be the number of times user i retweets the posts of user j in stage k, that is obtained as $\sum_{h=\{T,F\}} [\mathcal{N}_{ji}(\tau_{k+1}, h, \mathscr{R}) - \mathcal{N}_{ji}(\tau_k, h, \mathscr{R})]$ (from *Retweet MHP*). The observation $\mathbf{o}_{k,i}$ of a user i in stage k can then be represented as $\mathbf{o}_{k,i} = \{\mathbf{s}_{k,j}, (\{n_{\mathscr{R},k,l,j}, w_{k,l,j}\}_{l \in \mathbf{G}_{\cdot j}})\}_{j \in \mathbf{G}_{\cdot i}}$.

Reward At the end of each stage k, every user receives (from an external system) a reward $R_{M,k,i}$ defined as the correlation between the exposures to fake and true news among her Followers $\mathbf{G}_{i\cdot}$. We assume that this reward is provided to a user i

from the external system since the network is only locally/partially observable to the user and she cannot observe the state and/or actions of her Followers' other Followees $\{\{\cup \mathbf{G}_{.m}\}_{m\in\mathbf{G}_{i.}}/\{i\}\}$ (apart from i herself), that accounts for the number of times i's Followers are exposed to fake and true news. The total number of exposures to a user m in stage k is obtained as $\sum_{i\in\mathbf{G}_{.m}} n_{k,i}(h,z)$. Thus, the mitigation reward $R_{M,k,i}$ for user i at stage k is given as,

$$R_{M,k,i} = \sum_{z=\{\mathscr{T},\mathscr{R}\}} \frac{\sum_{m\in\mathbf{G}_{i.}} (\sum_{l\in\mathbf{G}_{.m}} n_{k,l}(T,z))(\sum_{l\in\mathbf{G}_{.m}} n_{k,l}(F,z))}{|\mathbf{G}_{.m}|} \tag{7.1}$$

Additionally, we consider a penalty $P_{k,i}$ for a user to post more, which corresponds to the cost or some amount of effort that a user needs to spend to post. While there can be different functions to quantify the penalty for posting more, we consider the penalty in this problem to be the number of posts made by a user as a fraction of the maximum number of posts made by her Followees.

$$P_{k,i} = \frac{\sum_{h=\{T,F\}} \sum_{z=\{\mathscr{T},\mathscr{R}\}} n_{k,i}(h,z)}{\max_{j\in\mathbf{G}_{.i}\cup i} \sum_{h=\{T,F\}} \sum_{z=\{\mathscr{T},\mathscr{R}\}} n_{k-1,j}(h,z)} \tag{7.2}$$

Thus, the total reward $R_{k,i}$ received by user i at stage k is given as

$$R_{k,i} = R_{M,k,i} - P_{k,i} \tag{7.3}$$

Hence, a user needs to learn a policy that mitigates the impact of fake news among her Followers, with fewer number of posts.

Objective The goal of each user i is to learn a policy $\pi_i : \mathbf{s}_{k,i} \rightarrow a_{\mathscr{T},k,i}, \{a_{\mathscr{R},k,j,i}\}_{j\in\mathbf{G}_{.i}}$ to determine her intervention actions for tweet and retweet processes for true news diffusion, in order to maximize the total expected discounted reward that she receives, for each stage k, i.e., $J_{k,i} = \sum_{k=1}^{K} \gamma^k \mathbb{E}[R_{k,i}]$, where $\gamma \in (0,1]$ is the discount rate. From Eq. 7.3, we can write,

$$\mathbb{E}[R_{k,i}] = \mathbb{E}[R_{M,k,i}] - \mathbb{E}[P_{k,i}] \tag{7.4}$$

From Eq. 7.1, we have,

$$\mathbb{E}[R_{M,k,i}] = \sum_{z=\{\mathscr{T},\mathscr{R}\}} \frac{\sum_{m\in\mathbf{G}_{i\cdot}}(\sum_{l\in\mathbf{G}_{\cdot m}}\mathbb{E}[n_{k,l}(T,z)])(\sum_{l\in\mathbf{G}_{\cdot m}}\mathbb{E}[n_{k,l}(F,z)])}{|\mathbf{G}_{\cdot m}|}$$

We assume that the diffusion of fake and true news is independent, and thus, we can decompose the expected reward due to these. For more details, see [15, 169]. Additionally, from Eq. 7.2, we compute,

$$\mathbb{E}[P_{k,i}] = \frac{\sum_{h=\{T,F\}}\sum_{z=\{\mathscr{T},\mathscr{R}\}}\mathbb{E}[n_{k,i}(h,z)]}{\max_{j\in\mathbf{G}_{\cdot i}\cup i}\sum_{h=\{T,F\}}\sum_{z=\{\mathscr{T},\mathscr{R}\}} n_{k-1,j}(h,z)} \tag{7.5}$$

For the intervention actions, we assume an upper cap on the amount of intervention for each user, i.e. $a_{z,k,i} \leq \rho$, where ρ. The interventions are continuous, and thus, the space of intervention actions for user i is given as $\mathbf{A}_{k,i} = \{a_{\mathscr{T},k,i}, a_{\mathscr{R},k,j,i} | a_{\mathscr{T},k,i} \in [0,\rho], a_{\mathscr{R},k,j,i} \in [0,\rho], j \in \mathbf{G}_{\cdot i}\}$.

7.5 DENPL Approach

Given the partially observable ego-network, the goal of a user is to learn a policy π_i that determines the intervention actions for her tweet and retweet intensities, based on her state and local observation to maximize her local reward. The local reward of a user depends on her Followers' exposures to fake and true news, which is impacted by the user's actions as well as the actions of other users (her Followers' Followees) whom she cannot observe. Thus, our key idea is to estimate the states of the Followers of a user, with the observation that a user serves both as a Followee and a Follower in her ego-network. Specifically, we learn a mapping between Followees' states and the user's (Follower) state, i.e., $f_{\mathcal{S}} : \{\mathbf{s}_{k,j}\}_{j\in\mathbf{G}_{\cdot i}} \rightarrow \mathbf{s}_{k,i}$, and then extrapolate it to estimate Followers' states from user's (Followee) state, i.e., $\{\mathbf{s}_{k,m}\}_{m\in\mathbf{G}_{i\cdot}} = f_{\mathcal{S}}(\mathbf{s}_{k,i})$. We propose partially centralized learning approach in which we only share the parameters between agents, without sharing the samples (i.e., state/actions/reward) between agents, in order to safeguard the privacy of the agents, by which we mean to limit data sharing

Algorithm 9 SequentialParameterSharing

Initialize $\boldsymbol{\theta}_\pi, \boldsymbol{\theta}_V, \boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im$ randomly.
repeat
  for $k = 1, ..., K$ **do**
    for $i = 1, ..., N$ **do**
      $\tilde{\boldsymbol{\phi}}_\mathcal{S}, \tilde{\boldsymbol{\phi}}_\mathcal{A}, \tilde{\boldsymbol{\theta}}_\mathcal{S}, \tilde{\boldsymbol{\theta}}_\mathcal{A}, \tilde{\boldsymbol{\theta}}_\Im$ = PolicyLearningPerUser($i, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_V, \boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im$)
      $\boldsymbol{\phi}_\mathcal{S}, \boldsymbol{\phi}_\mathcal{A}, \boldsymbol{\theta}_\mathcal{S}, \boldsymbol{\theta}_\mathcal{A}, \boldsymbol{\theta}_\Im = \tilde{\boldsymbol{\phi}}_\mathcal{S}, \tilde{\boldsymbol{\phi}}_\mathcal{A}, \tilde{\boldsymbol{\theta}}_\mathcal{S}, \tilde{\boldsymbol{\theta}}_\mathcal{A}, \tilde{\boldsymbol{\theta}}_\Im$
    end for
  end for
  for $i = 1, ..., N$ **do**
    /* Agent i updates policy based on her objective */
    $\tilde{\boldsymbol{\theta}_\pi}, \tilde{\boldsymbol{\theta}_V}$ = ComputeObjectiveUpdatePolicyForUser($i, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_V$)
    /* Set initial weights for the next user in the sequence */
    $\boldsymbol{\theta}_\pi, \boldsymbol{\theta}_V = \tilde{\boldsymbol{\theta}_\pi}, \tilde{\boldsymbol{\theta}_V}$
  end for
until $|\Delta\boldsymbol{\theta}_\pi| < \delta$ /* Convergence */
$\boldsymbol{\theta}^*_\pi = \boldsymbol{\theta}_\pi$
return $\boldsymbol{\theta}^*_\pi$

between agents. Thus, we use the same function for all users, i.e., $\pi_i = \pi$, however, the input and output corresponds solely to a single user. Our policy learning approach for a single user is presented in Algorithms 8-13, and we describe the components in detail next.

We represent the policy π as a function of user i's state, parameterized by weights $\boldsymbol{\theta}_\pi$, i.e., $a_{\mathscr{T},k,i}, \{a_{\mathscr{R},k,j,i}\}_{j \in \mathbf{G}_{.i}} = \pi(\mathbf{s}_k; \boldsymbol{\theta}_\pi)$. The value of user i's state is defined as the total expected reward when in the given state following policy π, i.e., $V_{k,i} = \mathbb{E}[\sum_{t=k}^{K} \gamma^t R_{t,i} | (\pi, \mathbf{s}_{k,i})]$. In a social network, due to peer-influence, the activities of a user affect the likelihood of activities of her Followers (Sec. 7.1). Thus, the value of a user's

Algorithm 10 ComputeObjectiveUpdatePolicyForUser($i, \boldsymbol{\theta}_\pi, \boldsymbol{\theta}_V$)

$L_{\theta,i} = 0, \quad L_{\phi,i} = 0$
for $k = 1, ..., K$ **do**
    Let $D_{k,i} = \sum_{j=k}^{K} \gamma^{k-1} r_{k,i}$
    $B_{k,i} = D_{k,i} - V_{k,i}$ /* Compute Advantage */
    $L_{\boldsymbol{\theta}_\pi,i} = L_{\boldsymbol{\theta}_\pi,i} + B_{k,i}$
    $L_{\boldsymbol{\theta}_V,i} = L_{\boldsymbol{\theta}_V,i} + ||V_{k,i} - D_{k,i}||_2$
end for
$J_{\theta,i} = L_{\boldsymbol{\theta}_\pi,i}, \quad J_{\boldsymbol{\theta}_V,i} = -L_{\boldsymbol{\theta}_V,i}$
/* Update parameters based only on user i's objective */
$\boldsymbol{\theta}_\pi = \boldsymbol{\theta}_\pi + \eta_{\boldsymbol{\theta}_\pi} \boldsymbol{\nabla}_{\boldsymbol{\theta}_\pi} J_{\boldsymbol{\theta}_\pi,i}, \quad \boldsymbol{\theta}_V = \boldsymbol{\theta}_V + \eta_{\boldsymbol{\theta}_V} \boldsymbol{\nabla}_{\boldsymbol{\theta}_V} J_{\boldsymbol{\theta}_V,i}$
return $\boldsymbol{\theta}_\pi$, $\boldsymbol{\theta}_V$

state is also conditioned on her ego-network, i.e., $V_{k,i} = \mathbb{E}[\sum_{t=k}^{K} \gamma^t R_{t,i} | (\pi, \mathbf{s}_{k,i}, \mathbf{G}_{.i}, \mathbf{G}_{i.})]$. It is computationally expensive to compute $V_{k,i}$ from future rewards for every possible policy π, and thus, we approximate the value as a function of the state of the user, the states of her Followees, and states of her Followers, parameterized by weights $\boldsymbol{\theta}_V$, i.e., $V_{k,i} = f_V(\mathbf{s}_{k,i}, \{\mathbf{s}_{k,j}\}_{j \in \mathbf{G}_{.i}}, \{\mathbf{s}_{k,m}\}_{m \in \mathbf{G}_{i.}}; \boldsymbol{\theta}_V)$. The total expected discounted reward can then be expressed as $J_{k,i} = \mathbb{E}[R_{k,i}] + \gamma\ f_V(\mathbf{s}_{k',i}, \{\mathbf{s}_{k',j}\}_{j \in \mathbf{G}_{.i}}, \{\mathbf{s}_{k',m}\}_{m \in \mathbf{G}_{i.}}; \boldsymbol{\theta}_V)$, where $\mathbf{s}_{k'}$ is the next state. We use advantage actor-critic algorithm (e.g., [28]) to learn continuous policies for agents since policy gradient methods are more effective in high dimensional spaces.

To realize partially centralized learning, we perform sequential weight update in which there is a random ordering of the users. A user i updates the weights based on her own objective i.e., $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \eta_{\boldsymbol{\theta}} \boldsymbol{\nabla}_{\boldsymbol{\theta}_\pi} J_{\boldsymbol{\theta},i}$, and then the next user j in the random ordering uses the initial weights as the weights learned by user i i.e. $\tilde{\boldsymbol{\theta}}$, and updates them based on her objective $J_{k,j}$. The algorithms for sequential parameter sharing and weight update are presented in Algorithms 9 and 10. To update the weights for functions π and V, for a single user, we compute $r_{k,i}$ that represents the expected reward obtained by user i for her actions $a_{\mathscr{T},k,i}, \mathbf{a}_{\mathscr{R},k,i}$ in stage k, i.e.,

Algorithm 11 LearnFolloweeFollowerDependency($i, \mathbf{s}_{k,i}, \mathbf{o}_{k,i}, \mathbf{a}_{z,k,i}, \{\mathbf{a}_{z,k-1,j}\}_{j\in\mathbf{G}_{.i}}, \boldsymbol{\phi}_{\mathcal{S}}, \boldsymbol{\phi}_{\mathcal{A}}, \boldsymbol{\theta}_{\mathcal{S}}, \boldsymbol{\theta}_{\mathcal{A}}, \boldsymbol{\theta}_{\Im}$)

repeat

/* Generalized representation for Followees' retweets, likes */

$\{(h^{\mathcal{A}}_{\mathscr{R},k-1,l,j}, h^{\mathcal{L}}_{k-1,l,j})\}_{j\in\mathbf{G}_{.i},l\in\cup\mathbf{G}_{.j}} = g_{\mathcal{A}}(\{(n_{\mathscr{R},k-1,l,j}, w_{k-1,l,j})\}_{j\in\mathbf{G}_{.i},l\in\cup\mathbf{G}_{.j}}; \boldsymbol{\phi}_{\mathcal{A}})$

/*Compute reconstruction loss for Followees' retweets, likes*/

$L^{\mathcal{A}}_{\phi} = \sum_{j\in\mathbf{G}_{.i}} ||\{h^{\mathcal{A}}_{\mathscr{R},k-1,l,j}\}_{l\in\cup\mathbf{G}_{.j}} - \{n_{\mathscr{R},k-1,l,j}\}_{l\in\cup\mathbf{G}_{.j}}||_2 + \sum_{j\in\mathbf{G}_{.i}} ||\{h^{\mathcal{L}}_{k-1,l,j}\}_{l\in\cup\mathbf{G}_{.j}} - \{w_{k-1,l,j}\}_{l\in\cup\mathbf{G}_{.j}}||_2$

/* Estimated retweets, likes of user i */

$\{\hat{n}_{\mathscr{R},k,j,i}, \hat{w}_{k,j,i}\}_{j\in\mathbf{G}_{.i}} = f_{\mathcal{A}}(\{(h^{\mathcal{A}}_{\mathscr{R},k-1,l,j}, h^{\mathcal{L}}_{k-1,l,j})\}_{j\in\mathbf{G}_{.i},l\in\cup\mathbf{G}_{.j}}; \boldsymbol{\theta}_{\mathcal{A}})$

/* Compute loss for estimated retweet actions of user i */

$L^{\mathcal{A}}_{i} = \sum_{j\in\mathbf{G}_{.i}} ||n_{\mathscr{R},k,j,i} - \hat{n}_{\mathscr{R},k,j,i}||_2 + \sum_{j\in\mathbf{G}_{.i}} ||w_{k,j,i} - \hat{w}_{k,j,i}||_2$

/* Generalized representation for user i's Followees' states */

$\{\mathbf{h}^{\mathcal{S}}_{k,j}\}_{j\in\mathbf{G}_{.i}} = g_{\mathcal{S}}(\{\mathbf{s}_{k,i}\}_{j\in\mathbf{G}_{.j}}; \boldsymbol{\phi}_{\mathcal{S}})$

/* Compute reconstruction loss $L^{\mathcal{S}}_{\phi}$ for Followees' states */

$L^{\mathcal{S}}_{\phi} = \sum_{j\in\mathbf{G}_{.i}} ||\mathbf{h}^{\mathcal{S}}_{k,j} - \mathbf{s}_{k,j}||_2$

/* Attention between user i and Followees based on state */

$\alpha^{\mathcal{S}}_{k,j,i} = \sigma(\Im(\mathbf{s}_{k-1,i}, \mathbf{h}^{\mathcal{S}}_{k,j}; \theta_{\Im}))$

/* Estimate state of user i */

$\hat{\mathbf{s}_{k,i}} = f_{\mathcal{S}}(\mathbf{s}_{k-1,i}, \sum_{j\in\mathbf{G}_{.i}} \alpha^{\mathcal{S}}_{k,j,i}\, \mathbf{h}^{\mathcal{S}}_{k,j}, \{\hat{n}_{\mathscr{R},k,j,i}, \hat{w}_{k,j,i}\}_{j\in\mathbf{G}_{.i}}; \theta_{\mathcal{S}})$

/* Compute loss $L^{\mathcal{S}}_{i}$ for estimated state of user i */

$L^{\mathcal{S}}_{i} = ||\mathbf{s}_{k,i} - \hat{\mathbf{s}}_{k,i}||_2$

$L_i = L^{\mathcal{S}}_{i} + L^{\mathcal{A}}_{i} + L^{\mathcal{S}}_{\phi} + L^{\mathcal{A}}_{\phi}$ /* Total Loss */

/* Update Weights for user i */

$\boldsymbol{\phi}_{\mathcal{S}} = \boldsymbol{\phi}_{\mathcal{S}} + \eta_{\phi_{\mathcal{S}}} \boldsymbol{\nabla}_{\phi_{\mathcal{S}}} L_i(\boldsymbol{\phi}_{\mathcal{S}})$, $\boldsymbol{\phi}_{\mathcal{A}} = \boldsymbol{\phi}_{\mathcal{A}} + \eta_{\phi_{\mathcal{A}}} \boldsymbol{\nabla}_{\phi_{\mathcal{A}}} L_i(\boldsymbol{\phi}_{\mathcal{A}})$, $\boldsymbol{\theta}_{\mathcal{A}} = \boldsymbol{\theta}_{\mathcal{A}} + \eta_{\theta_{\mathcal{S}}} \boldsymbol{\nabla}_{\theta_{\mathcal{S}}} L_i(\boldsymbol{\theta}_{\boldsymbol{\mathcal{S}}})$, $\boldsymbol{\theta}_{\mathcal{A}} = \boldsymbol{\theta}_{\mathcal{A}} + \eta_{\theta_{\mathcal{A}}} \boldsymbol{\nabla}_{\theta_{\mathcal{A}}} L_i(\boldsymbol{\theta}_{\boldsymbol{\mathcal{A}}})$, $\boldsymbol{\theta}_{\Im} = \boldsymbol{\theta}_{\Im} + \eta_{\theta_{\Im}} \boldsymbol{\nabla}_{\theta_{\Im}} L_i(\boldsymbol{\theta}_{\Im})$

until $|\Delta\boldsymbol{\theta}_{\mathcal{S}} < \delta|$ /* Convergence */

return $\boldsymbol{\phi}_{\mathcal{S}}, \boldsymbol{\phi}_{\mathcal{A}}, \boldsymbol{\theta}_{\mathcal{S}}, \boldsymbol{\theta}_{\mathcal{A}}, \boldsymbol{\theta}_{\Im}$

$r_{k,i} = \mathbb{E}_i[R_{k,i}] + \gamma V_{k,i}$. Let $D_{k,i}$ be the total discounted reward for stage k, where K is the total number of stages, i.e., $D_{k,i} = \sum_{j=k}^{K} \gamma^k r_{k,i}$ The parameters $\boldsymbol{\theta}_V$ of the value function are updated with the loss $||V_{k,i} - D_{k,i}||_2$, as the value function corresponds to the total expected discounted reward that a user can obtain in a given state. To update the policy function parameters $\boldsymbol{\theta}_\pi$, we optimize the advantage function that is obtained by subtracting $V_{k,i}$ as baseline from $D_{k,i}$, which helps to compute the relative advantage of actions by subtracting an average amount of return, and also reduce the variance. Note that, to avoid the problem of non-stationary policies due to individual learning by each agent [173], we consider that agents perform updates in a sequence, and the actions of agents in stage k are updated after the actions of all agents have been updated in stage $k-1$.

Moreover, each user has different number of Followees and Followers in her ego-network, i.e., the dimensions of input and output while extrapolation using $f_{\mathcal{S}}$, are different from those while learning $f_{\mathcal{S}}$. Thus, we consider that the dimensionality of input and output of shared oracle (neural network) corresponds to the total number of users N, and that the input and output are sorted by user ids. While learning the function with respect to a given user i, we use a sentinel to receive the signal only from the Followees of the user, and determine the output only for the Followers of the user. We use the same approach to learn different functions $\pi, V, f_{\mathcal{S}}, f_{\mathcal{A}}, g_{\mathcal{S}}, g_{\mathcal{A}}$.

Now, to update the parameters for policy π at stage k, we need to accurately estimate the total expected discounted reward from future stages $J_{k,i}$, that depends on the estimates of the value function $V_{k,i}$. However, to approximate the value function for a user, we require the state of her Followers, which is hidden from her partial view of the environment. Thus, we learn a mapping between the state of Followees and user's state, and then use the learned mapping to extrapolate the states of her Followers using her state and local observation.

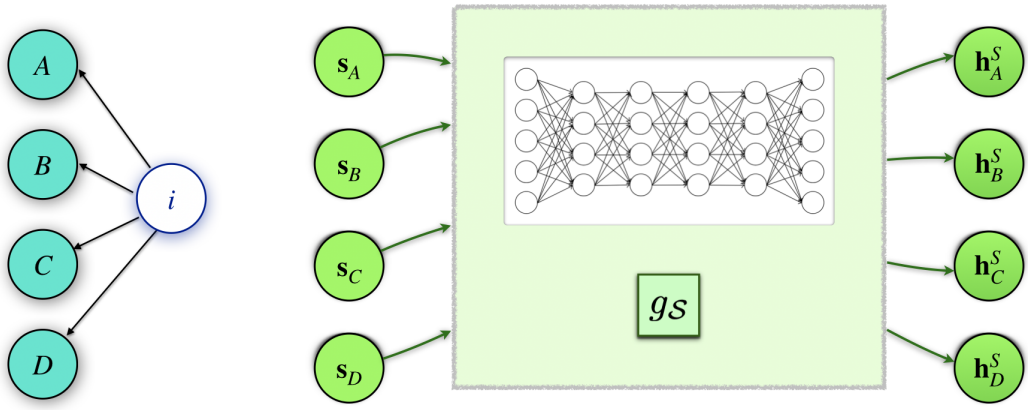


Figure 7.3.: Learn generalized representation of Followees' states

7.5.1 Learning dependency between Followees-Followers states

Each user i learns a function $f_{\mathcal{S}}$ parameterized by weights $\boldsymbol{\theta}_{\mathcal{S}}$ to estimate her state $\hat{\mathbf{s}}_{k,i}$ from the states of her Followees $\{\mathbf{s}_{k,j}\}_{j \in G_{.i}}$, with an objective $L_i^{\mathcal{S}}$ to minimize the difference between the estimated and true states of the user, as shown in line 19, Alg. 11. We first obtain a generalized representation for the Followees' states using stacked auto-encoders (e.g., [174]) so that the same function that is used to estimate user i's state from her Followees' states, can be used to estimate user i's Followers' states from user i's state, i.e., the input can be generalized across different sets of users. Specifically, user i learns a function $g_{\mathcal{S}} : \mathbb{R}^{|\mathbf{G}_{.i}|d} \rightarrow \mathbb{R}^{|\mathbf{G}_{.i}|d}$, parameterized by weights $\boldsymbol{\phi}_{\mathcal{S}}$ (where $|\mathbf{G}_{.i}|$ is the number of Followees of user i), to obtain the generalized representation $\{\mathbf{h}_{k,j}^{\mathcal{S}}\}_{j \in \mathbf{G}_{.i}}$ for the states of her Followees as shown in line 11, Alg. 11 with reconstruction loss $L_{\phi}^{\mathcal{S}}$ in line 13, Alg. 11. Fig. 7.3 shows the input and output for the model g_S using an example of user's partially observable ego-network in Fig. 7.1. To improve the estimates for the Followers' states, it is important to consider the attention that a Follower pays to her Followees to account for the different peer-influence between users.

7.5.2 Attention between users based on State

A user i learn the attention weights $\alpha^{\mathcal{S}}_{k,j,i}$ between her state in the past stage $\mathbf{s}_{k-1,i}$, and the generalized state of her Followee j in the current stage k, using an attention mechanism $\Im : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$. The attention weights are computed as $\alpha^{\mathcal{S}}_{k,j,i} = \sigma(\Im(\mathbf{s}_{k-1,i}, \mathbf{h}^{\mathcal{S}}_{k,j}; \theta_{\Im}))$ (line 14, Alg. 11), where σ is the sigmoid function. Similar to [175], the attention mechanism $\Im$ is learned using a multi-layer feed-forward neural network, in which the gradient of the total loss is back-propagated through to jointly learn the weights for the attention mechanism along with the total loss for user i, $L_i = L^{\mathcal{S}}_i + L^{\mathcal{A}}_i + L^{\mathcal{S}}_\phi + L^{\mathcal{A}}_\phi$ (line 18, Alg. 11), where $L^{\mathcal{A}}_i$ and $L^{\mathcal{A}}_\phi$ are described later in Sec. 7.5.3.

Each user has different number of Followees and Followers. Therefore, the mapping between Followees' states and a user i's state is the mapping from $\mathbb{R}^{|\mathbf{G}_{.i}|d}$ to $\mathbb{R}^d$, where d is the number of network activities (Sec. 7.3). On the agent level, it is a many-to-one mapping from the Followees to the user, i.e., $|\mathbf{G}_{.i}| \rightarrow 1$. Now, if user i uses the same mapping to extrapolate the states of her Followers by providing her state as the input, the mapping will be $\mathbb{R}^d \rightarrow \mathbb{R}^{|\mathbf{G}_{i.}|d}$, that, on the agent level, is a one-to-many mapping from the state of the agent to the states of her Followers, i.e., $1 \rightarrow |\mathbf{G}_{i.}|$. Hence, the estimates of the states of the Followers of user i computed using user i's state and attention weights between user i and her Followers will be less accurate. To address this, we utilize the pairwise interactions, i.e., the retweets and likes that a user provides to her Followees, to learn a many-to-many mapping that can be used effectively to extrapolate the states of the Followers, described as follows.

7.5.3 Learning dependency between Followees-Followers retweets and likes (actions)

A user i learns a mapping from the number of retweets and likes (defined in Sec. 7.4.3) given by her Followees (to their Followees), to the number of retweets and likes given by her (as a Follower) to her Followees, and then uses the same mapping to extrapolate the retweets and likes given by her Followers to herself (acting

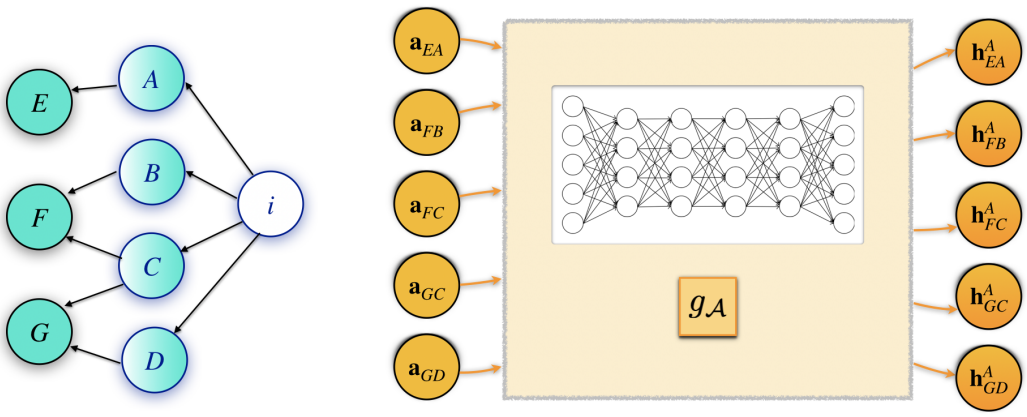


Figure 7.4.: Learn generalized representation of Followees' actions

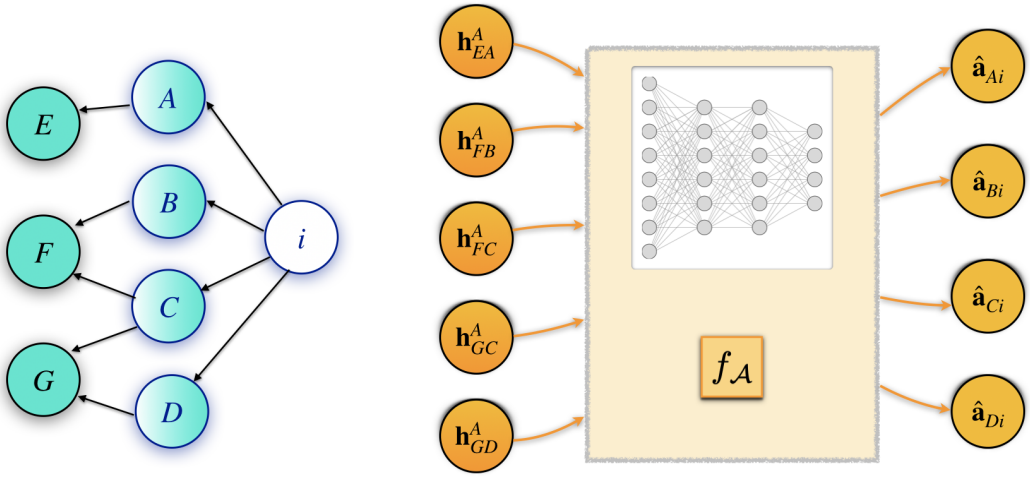


Figure 7.5.: Learn to estimate user's actions for Followees

as a Followee). First, we generalize the input representation across different sets of users using stacked auto-encoders, to effectively transfer this mapping. Each user i learns a function $g_{\mathcal{A}}$, parameterized by weights $\boldsymbol{\phi}_{\mathcal{A}}$, to obtain the generalized representation $\{(h^{\mathcal{A}}_{\mathscr{R},k-1,l,j}, h^{\mathcal{L}}_{k-1,l,j})\}_{j\in\mathbf{G}_{.i},l\in\cup\mathbf{G}_{.j}}$ for the number of retweets and likes given by her Followees, as shown in line 3, Alg. 11 with reconstruction loss $L^{\mathcal{A}}_{\phi}$ in line 5, Alg. 11. Fig. 7.4 shows the input and output for the model g_A using an example of user's partially observable ego-network. Then, we learn a function $f_{\mathcal{A}}$ parameterized by weights $\boldsymbol{\theta}_{\mathcal{A}}$, to estimate the number of retweets and likes given by a user to her Followees $\{\hat{n}_{\mathscr{R},k,j,i}, \hat{w}_{k,j,i}\}_{j\in\mathbf{G}_{.i}}$ from the generalized representation of the number of

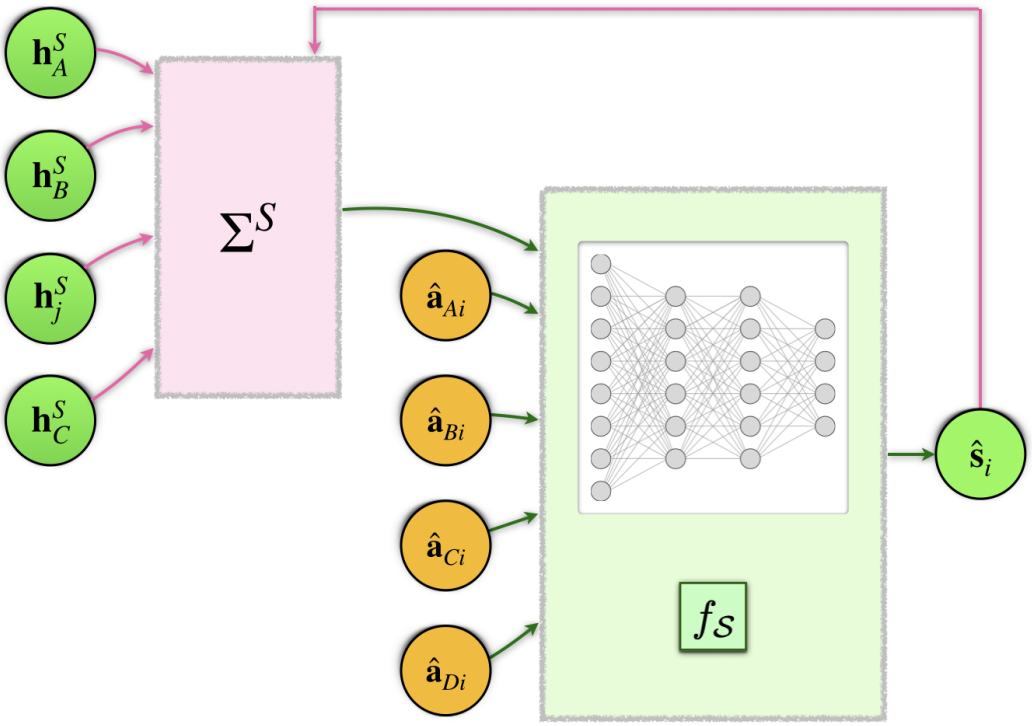


Figure 7.6.: Learn to estimate user's state from Followees' states

retweets and likes of her Followees as shown in line 7, Alg. 11. The parameters $\boldsymbol{\theta}_{\mathcal{A}}$ are learned to minimize $L_i^{\mathcal{A}}$ which is the difference between the estimated and actual number of retweets/likes for the user, as shown in line 9, Alg. 11. Fig. 7.5 shows the input and output for the model f_A using an example of user's partially observable ego-network. The estimated number of retweets and likes of a user, together with the user's previous state in stage $k-1$ and the state of her Followees are then provided as input to $f_{\mathcal{S}}$ to estimate the user's state in stage k, as shown in line 17, Alg. 11. Fig. 7.6 shows the input and output for the model f_S using an example of user's partially observable ego-network.

7.5.4 Extrapolating Followers' states

After learning $f_{\mathcal{S}}$, we use it to extrapolate the states of a user i's Followers described as follows. We propose to use the states of user i's Followees, along with user i's state, to improvise the estimates of user i's Followers' states. We first ob-

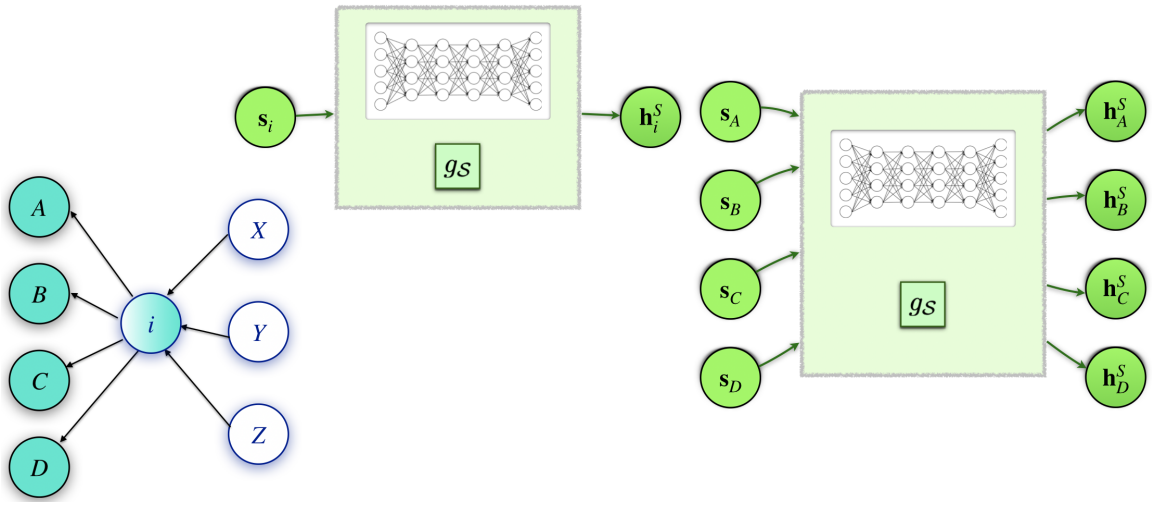


Figure 7.7.: Obtain generalized representation of user's state and Followees' states

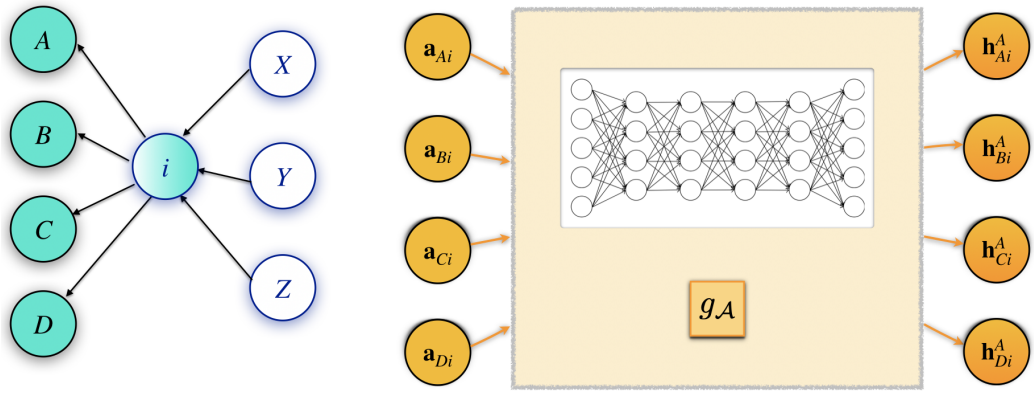


Figure 7.8.: Obtain generalized representation of user's actions

tain the generalized representations $\mathbf{h}_{k,i}^{\mathcal{S}}$ and $\{\mathbf{h}_{k,j}^{\mathcal{S}}\}_{j \in \mathbf{G}_{.i}}$, respectively, for the states of user i and her Followees using the learned function $g_{\mathcal{S}}$ (Fig. 7.7), as shown in line 2, Alg. 12. Next, our idea is to utilize the attention weights between the retweets/likes of user i's Followers (for user i), and the retweets/likes of a user i (for user i's Followees), to capture the indirect dependency between user i's Followees and user i's Followers. Specifically, the user utilizes the learned function $g_{\mathcal{A}}$ to obtain the generalized representation for the number of retweets and likes provided by her to her Followees (Fig. 7.8), which is then used to extrapolate the number of retweets and likes given by her Followers using learned function $f_{\mathcal{A}}$ (Fig. 7.9), as shown in

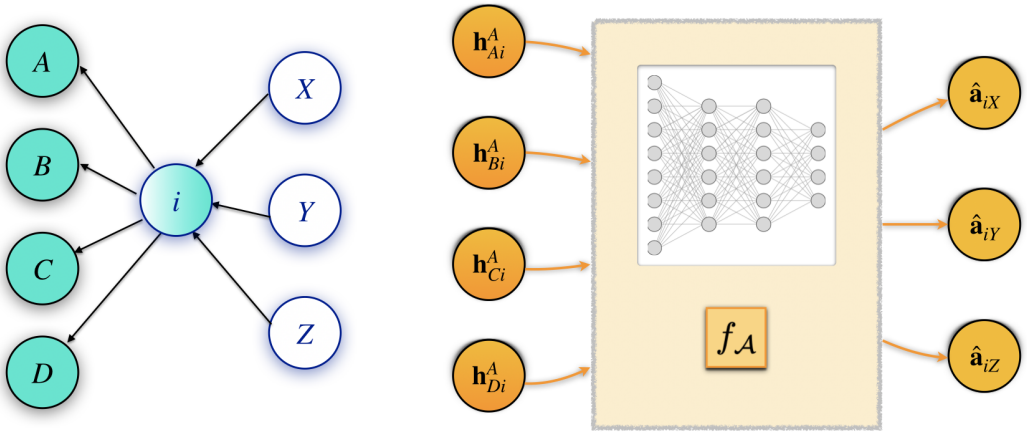


Figure 7.9.: Estimate Followers' actions for user

Alg. 13. The attention that a user i's Follower m provides to the user's Followee j, in stage k, is $\alpha^{\mathcal{A}}_{k,j,m} = \theta_{\mathcal{A}}[h^{\mathcal{A}}_{\mathscr{R},k-1,j,i}, \hat{n}_{\mathscr{R},k,i,m}] + \theta_{\mathcal{A}}[h^{\mathcal{L}}_{k-1,j,i}, \hat{a}_{\mathscr{R},k,i,m}]$ (line 6, Alg. 13) where $\theta_{\mathcal{A}}[h^{\mathcal{A}}_{\mathscr{R},k-1,j,i}, \hat{n}_{\mathscr{R},k,i,m}]$ is the weight between the node corresponding to the representation for number of retweets of user i for her Followee j, and the output node that corresponds to the estimated number of retweets of Follower m for user i, in the neural network. Similarly, $\theta_{\mathcal{A}}[h^{\mathcal{L}}_{k-1,j,i}, \hat{a}_{\mathscr{R},k,i,m}]$ corresponds to number of likes. Then, using the learned $f_{\mathcal{S}}$ we estimate the state of user i's Follower m, in stage k (Fig. 7.10), as a function of the estimated state of m in stage $k-1$, the generalized state representation of user i weighted by $\alpha^{\mathcal{S}}$ (Sec. 7.5.2), and the generalized state representation of user i's Followees weighted by $\alpha^{\mathcal{A}}$ as shown in line 6, Alg. 12.

Note that, we capture the impact of two-hop Followees of m by learning the attention weights $\alpha^{\mathcal{A}}_{k,j,m}$ based only on the actions of one-hop Followee i.e., user i. With this, we also avoid storing the explicit states/actions of the two-hop Followees to obtain the attention provided to them by the Follower m. Additionally, we randomly sampled separate set of Followees for learning and separate set for extrapolation, and did not find much difference compared to using the complete set of a user's Followees for both learning and extrapolation. Hence, the latter case does not bias our results.

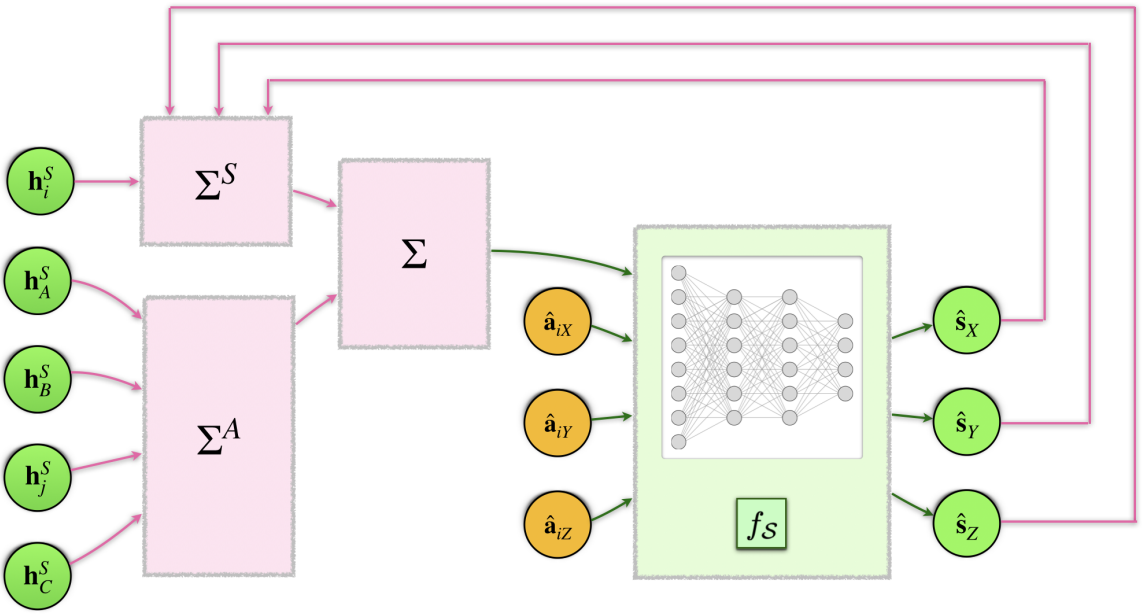


Figure 7.10.: Estimate Followers' states

Algorithm 12 ExtrapolateFollowersStates($i, \mathbf{s}_{k,i}, \{\mathbf{s}_{k,j}\}_{j\in\mathbf{G}_{.i}}, \{(\hat{n}_{\mathscr{R},k,i,m}, \hat{w}_{k,i,m}\}_{m\in\mathbf{G}_{i.}}, \{\alpha^{\mathcal{A}}_{k,j,m})\}_{m\in\mathbf{G}_{i.}, j\in\mathbf{G}_{.i}}, \boldsymbol{\phi}_{\mathcal{S}}, \boldsymbol{\theta}_{\mathcal{S}}, \boldsymbol{\theta}_{\Im}$)

1: /* Generalized representation for state of user i and her Followees */

2: $\mathbf{h}^{\mathcal{S}}_{k,i} = g_{\mathcal{S}}(\mathbf{s}_{k,i}; \boldsymbol{\phi}_{\mathcal{S}})$, $\{\mathbf{h}^{\mathcal{S}}_{k,j}\}_{j\in\mathbf{G}_{.i}} = g_{\mathcal{S}}(\{\mathbf{s}_{k,j}\}_{j\in\mathbf{G}_{.i}}; \boldsymbol{\phi}_{\mathcal{S}})$

3: /* Attention between Followers and user i based on state */

4: $\alpha^{\mathcal{S}}_{k,i,m} = \sigma(\Im(\hat{\mathbf{s}}_{k-1,m}, \mathbf{h}^{\mathcal{S}}_{k,i}; \boldsymbol{\theta}_{\Im}))$

5: /* Estimated Followers' states */

6: $\hat{\mathbf{s}}_{k,m} = f_{\mathcal{S}}(\hat{\mathbf{s}}_{k-1,m}, \alpha^{\mathcal{S}}_{k,i,m}\ \mathbf{h}^{\mathcal{S}}_{k,i} + \sum_{j\in\mathbf{G}_{.i}} \alpha^{\mathcal{A}}_{k,j,m}\ \mathbf{h}^{\mathcal{S}}_{k,j}, \{(\hat{n}_{\mathscr{R},k,i,m}, \hat{w}_{k,i,m}\}_{m\in\mathbf{G}_{i.}}; \boldsymbol{\theta}_{\mathcal{S}})\quad \forall m \in \mathbf{G}_{i.}$

7: **return** $\{\hat{\mathbf{s}}_{k,m}\}_{m\in\mathbf{G}_{i.}}$

After extrapolating the Followers' states, user i can estimate her value function $V_{k,i}$ based on her own state, the states of her Followees, and the estimated states of her Followers, i.e., $V_{k,i} = f_V(\mathbf{s}_{k,i}, \{\mathbf{s}_{k,j}\}_{j\in\mathbf{G}_{.i}}, \{\mathbf{s}_{k,m}\}_{m\in\mathbf{G}_{i.}}; \boldsymbol{\theta}_V)$, (line 9, Alg. 8).

Algorithm 13 ExtrapolateFollowersRetweetsLikes($i, \{a_{\mathscr{R},k-1,j,i}\}_{j\in\cup\mathbf{G}_{.i}}, \boldsymbol{\phi}_{\mathcal{A}}, \boldsymbol{\theta}_{\mathcal{A}}$)

/* Generalized representation for user i's retweets, likes */

$\{(h^{\mathcal{A}}_{\mathscr{R},k-1,j,i}, h^{\mathcal{L}}_{k-1,j,i})\}_{j\in\cup\mathbf{G}_{.i}} = g_{\mathcal{A}}(\{(n_{\mathscr{R},k-1,j,i}, w_{k-1,j,i})\}_{j\in\cup\mathbf{G}_{.i}}; \boldsymbol{\phi}_{\mathcal{A}})$

/* Estimated retweet, likes of user i's Followers */

$\{\hat{n}_{\mathscr{R},k,i,m}, \hat{w}_{k,i,m}\}_{m\in\mathbf{G}_{i.}} = f_{\mathcal{A}}(\{(h^{\mathcal{A}}_{\mathscr{R},k-1,j,i}, h^{\mathcal{L}}_{k-1,j,i})\}_{j\in\mathbf{G}_{.i}}; \boldsymbol{\theta}_{\mathcal{A}})$

/*Attention between Followees and Followers based on retweets, likes*/

$\alpha^{\mathcal{A}}_{k,j,m} = \theta_{\mathcal{A}}[h^{\mathcal{A}}_{\mathscr{R},k-1,j,i}, \hat{n}_{\mathscr{R},k,i,m}] + \theta_{\mathcal{A}}[h^{\mathcal{L}}_{k-1,j,i}, \hat{a}_{\mathscr{R},k,i,m}] \quad \forall j \in \mathbf{G}_{.i}, \forall m \in \mathbf{G}_{i.}$

return $\{(\hat{n}_{\mathscr{R},k,i,m}, \hat{w}_{k,i,m})\}_{m\in\mathbf{G}_{i.}}, \{\alpha^{\mathcal{A}}_{k,j,m}\}_{m\in\mathbf{G}_{i.}, j\in\mathbf{G}_{.i}}$

7.5.5 Policy Evaluation

For each user i, we find the intervention actions $\mathbf{a}_{z,k,i} = \pi(\mathbf{s}_{k,i}; \boldsymbol{\theta}_{\pi})$ given her state $\mathbf{s}_{k,i}$ that is obtained from events in the evaluation data. To obtain the actions, we do not need to estimate the Followers' states as π is conditioned only on the state of user i. Then, we simulate the MHPs after adding $\mathbf{a}_{z,k,i}$ to the respective base exogenous intensities of tweet ($z = \mathscr{T}$) and retweet ($z = \mathscr{R}$) processes for true news diffusion, and compute the following evaluation metric. To compare with previous work [43], we compare the performance of different methods based on the *collective reward* obtained across all users in the network. Let performance $\mathcal{P} = \sum_{k=1}^{K} R_k \times \frac{|L_{T,k} \cap L_{F,k}|}{|L_{F,k}|}$, where $L_{T,k} = \{i | i \in [1, N], \mathbf{n}_k(T, z) \cdot \mathbf{G}_{.i} > 0\}$ and $L_{F,k} = \{i | i \in [1, N], \mathbf{n}_k(F, z) \cdot \mathbf{G}_{.i} > 0\}$ are the sets of users exposed to true and fake news, respectively, during stage k, and R_k is the collective reward defined as the correlation between exposures to fake and true news across all users in the network. To evaluate the performance of different methods based on individual rewards, we compute the reward $R_{k,i}$ for a user which is the correlation between exposures to fake and true news among the Followers of user i along with the fraction of the Followers exposed to fake news that become exposed to true news, that helps to assign more importance to the selection of distinct users over the selection of few users with high exposures. Then, we compare different methods based on the *cumulative reward*, i.e., $\sum_{k=1}^{K} \sum_{i=1}^{N} R_{k,i} \times \frac{|L_{T,k,i} \cap L_{F,k,i}|}{|L_{F,k,i}|}$, where $L_{T,k,i} = \{i | i \in \mathbf{G}_{i.}, \mathbf{n}_k(T, z) \cdot \mathbf{G}_{.i} > 0\}$ and $L_{F,k,i} = \{i | i \in \mathbf{G}_{i.}, \mathbf{n}_k(F, z) \cdot \mathbf{G}_{.i} > 0\}$ are

the sets of Followers of user i exposed to true and fake news, respectively. To assess intervention gains, we measure the performance relative to that obtained by applying no intervention. Specifically, we compute the difference between the performance after applying the learnt policy and that without applying a policy.

7.6 Experiments

We use real-world datasets, Twitter 2016 and Twitter 2015 [5, 146], with 750 and 2050 users, respectively. The time-horizon $\mathcal{T} = 40$ is divided into 40 stages of $\Delta T = 1$ hour each. The time-intervals for Training Data, STD, SED before and after applying interventions, and Held-Out Data (Fig. 7.2), respectively, are, $[0, 10)$, $[10, 20)$, $[20, 30)$, and $[30, 40)$. We let $\rho_k \sim \mathcal{U}(0, 1)$, and $\gamma = 0.7$, as in [15, 169].

We consider the baselines listed next to compare the performance of our approach. Note that we perform sequential parameter sharing in all the baselines, except DEC, due to the large number of users in the network. To compare the performance with respect to N, we down-sample the network. Specifically, we select a user i randomly. Then we select atmost $\zeta \leq 5$ Followees and Followers of the user randomly. We then repeat this process for all the selected users until we reach the desired network size limit. For each network size, we down-sample 20 different networks and report average results.

7.6.1 Baselines

OS (Observed State)

An agent learns the value function based only on her state and the state of her Followees (e.g., [93]).

TFS (True Follower States)

To compare to the scenario where agents would have access to the true state of their Followers (but not to the complete network state of all users), we consider this baseline in which an agent uses the true state of her Followers along with her state and Followees' states to approximate the value function.

EFS (Extrapolated Follower State)

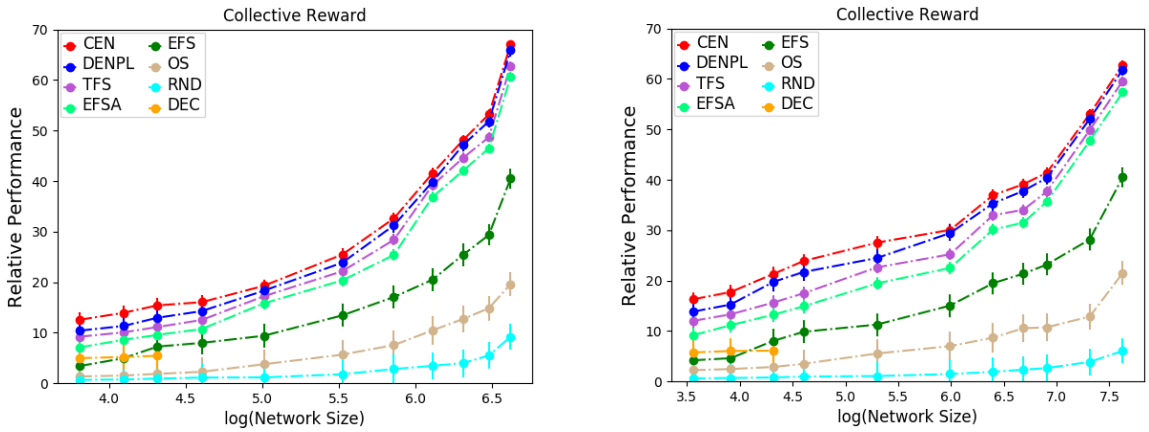
This version of our proposed extrapolated modeling approach learns a many-to-one mapping from the state of a user's Followees to the user's state, and then transfers the model to estimate user's Followers' states from user's state (one-to-many mapping). Agent's state, Followees' states and the estimated Followers' states are used to approximate the value function.

EFSA (Extrapolated Follower States + Actions)

This version of our proposed approach which performs extrapolated modeling via many-to-many mapping learned using the retweet actions and likes between users, but the state of user's Followees is not used to estimate the states of her Followers. Agent's state, observation and the estimated Followers' states are used to approximate the value function.

CEN (Fully Centralized Learning)

The environment is fully observable to all agents, and they observe the complete network state, and learn actions to maximize their individual rewards. The actions are conditioned on the complete network state, as opposed to TFS, in which the actions are conditioned only on the state of a user. The complete network state is used to approximate the value function (e.g., [15, 43, 71]).



(a) Twitter 2016 (b) Twitter 2015

Figure 7.11.: Performance vs N (Collective Reward)

DEC (Fully Decentralized Learning)

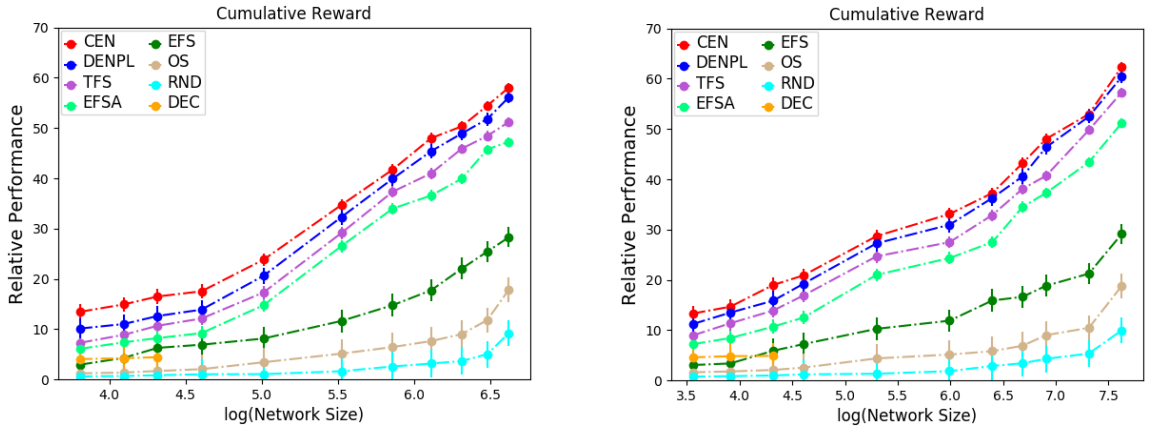
The components are same as in DENPL, however, a separate model is learned for each user, without sharing any parameters (for small number of users) (e.g., [44, 107, 119]).

RND (Random Actions)

Random. $a_{z,k,i} \propto \mathcal{U}(0, \rho)$.

7.6.2 Results

We compare the relative performance of different methods with respect to network size N, based on Collective Reward in Fig. 7.11, and Cumulative Reward in Fig. 7.12. We find that the results for both are qualitatively the same. Our proposed approach performs equivalent to the centralized learning method CEN, and outperforms other baselines, for all network sizes. We observe that our method DENPL, TFS, EFS, and EFSA that use sequential parameter sharing described in Sec. 7.5, achieve a much higher performance and lower variance compared to the fully decentralized



(a) Twitter 2016 (b) Twitter 2015

Figure 7.12.: Performance vs N (Cumulative Reward)

method DEC. This is because the effective number of samples per user increases due to sharing parameters. Thus, by only sharing the parameters that are updated individually by each user based on her own objective, without sharing any actual data/samples between users, we can achieve large gains in performance with lower variance.

We note that there is very little change in the performance of the fully decentralized method DEC with respect to the number of agents N, and the results have high variance. This is due to the fact that each agent learns her policy independently, conditioned solely on her local neighborhood, without sharing any parameters or trajectory information from other agents in the network. Hence, there are less samples available to improvise the estimates for each user. Also, the memory requirements increase with an increase in the number of agents as we need to store separate set of parameters for every agent. This shows that fully decentralized learning does not yield accurate policy estimates, and adds extra sample complexity to the task, due to insufficient information per user. On the other hand, the performance of partially centralized learning methods decreases with a decrease in N. This shows that with parameter sharing, the sample efficiency improves with an increase in the number

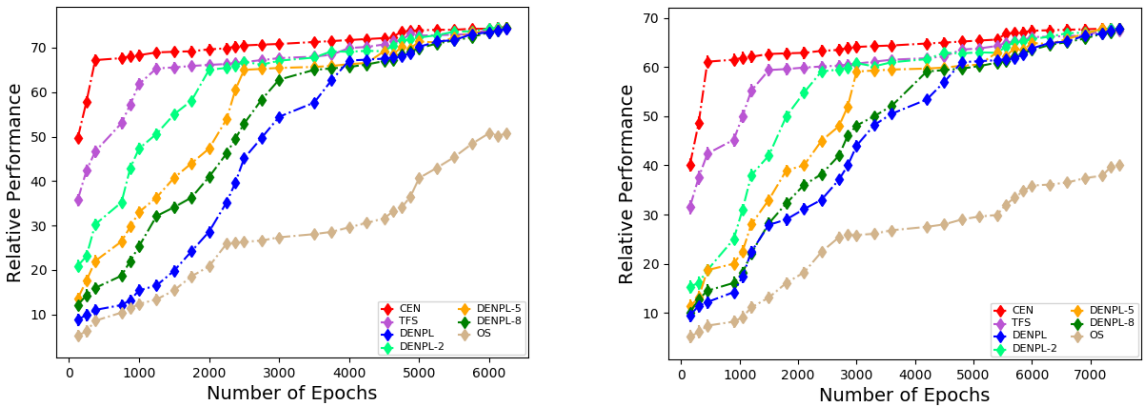
of agents, as the samples of more users are available to optimize the parameters, resulting in better performance.

We observe a large difference in the performance of method OS in which a user simply learns the value function based on her state and her Followees' states, and method TFS, in which the user also includes the true states of her Followers to approximate the value function. This shows that it is important to use the states of the Followers to learn the value function. In practice, the user cannot observe her Followers' states, and therefore it is important to estimate the Followers' (hidden) states and use it for policy learning.

The method EFS that estimates the states of the Followers and utilize those to approximate the value function, outperforms OS. This indicates that including the estimates of Followers' states via ego-network extrapolation helps to improvise the policy. However, the performance of EFS is still lower compared to TFS, which indicates that simply inverting a many-to-one mapping between Followees and a user, to a one-to-many mapping between the user and Followers does not yield accurate estimates of Followers' states. This is due to insufficient input signals while estimating the states of Followers from the state of a user alone.

The method EFSA, which uses the pairwise user interactions to estimate the Followers' retweets and likes and in-turn use those for estimating the Followers' states, achieves a much higher performance than EFS, and is closer to TFS. Thus, pairwise user interactions help to effectively extrapolate a many-to-many mapping.

Our proposed approach DENPL performs better than EFSA and TFS. This indicates that along with the estimated Followers' retweets/likes, utilizing the state of the Followees combined with the attention learned between the Followees and Followers of a user, helps to incorporate information about two-hop (more than one-hop) Followees, and thus obtain better estimates for Followers' states. Our method achieves performance equivalent to the centralized learning method CEN, that has the highest performance since it assumes a fully observable environment and learns conditioned on the complete network state of all the agents, whereas other approaches consider a



(a) Twitter 2016

(b) Twitter 2015

Figure 7.13.: Convergence

partially observable environment. However, fully centralized learning is computationally intensive due to high dimensional joint state/action representations that grows larger with the number of agents. It is important to note that we cannot perform clustering as in Chapter 6, since that requires a centralized controller that has access to the states/actions/rewards of all users for clustering, and thus, clustering will be performed at a global level. However, we want to learn in a more decentralized fashion to safeguard privacy by not sharing the data between users. This indicates that by effectively extrapolating the dependencies learned from the Followees to Followers using pairwise user interactions and peer-influence as attention, we are able to learn policies equivalent to those learned via centralized learning. In addition, by performing sequential parameter sharing, we are able to improve the sample efficiency, and also ensure that the policies are more privacy-aware (limited data-sharing between users) (Sec. 7.1).

We also conducted additional experiments to study the challenge, in a partially observable setting, due to the unavailability of the Followers' states, as compared to the scenario where the hidden state becomes available after a few time-steps (i.e. before the time-horizon). In the latter, the true state can be utilized as history for policy learning, however, in the former, as in our problem setting, the hidden state is

Table 7.1.: Sum of Retweets at $\tau + g$ for Users with Interventions at τ

MODEL	$\tau' = \tau + 0$		$\tau' = \tau + 2$		$\tau' = \tau + 5$		$\tau' = \tau + 8$	
	I	$\neg I$	I	$\neg I$	I	$\neg I$	I	$\neg I$
CEN	**956.5**	**410.8**	**775.4**	**229.3**	**625.8**	**210.8**	**560.2**	**177.7**
DENPL	**938.9**	**418.1**	**765.8**	**239.3**	**610.2**	**230.6**	532.9	**182.6**
TFS	925.5	426.7	764.7	245.9	599.8	245.6	**552.8**	192.2
EFSA	849.7	544.8	712.9	323.8	555.1	301.4	498.5	239.7
EFS	597.6	565.7	554.8	361.8	519.7	331.4	433.7	301.3
OS	492.3	426.2	430.6	368.1	331.9	302.6	282.7	233.3
RND	400.6	515.8	258.1	414.6	338.6	349.4	112.8	376.8

not available to agents for learning (even beyond the time-horizon), due to the directed nature of user interactions. In Fig. 7.13, we compare the time until convergence required by different methods. The methods CEN, TFS assume the availability of the true state of Followers, and hence converge much faster compared to other methods. On the other hand, our approach DENPL and OS do not have the state information of the Followers available, and hence these take longer to learn accurate state estimates. We also compared with the versions of our approach, DENPL-2, DENPL-5, DENPL-8, where we assume that the state information becomes available after 2, 5, 8 time-steps, respectively, and use that as history to improvise the estimates. These versions converge faster, but as the number of time-steps (when true state information becomes available) increases, it takes longer to converge. Recall however, this requires storing the trajectory for every user, which is space inefficient for social networks.

The above results, based on reward, serve as proof of concept that increasing the intensity of posting true news by a user helps to mitigate the impact of fake news locally, i.e., increase the exposure of true news, among the Followers who are exposed to fake news. Since we cannot make real-time interventions, we compare different methods by measuring the impact of users who increase their intensity for true news

diffusion, in terms of the total number of users actually reached in the real Held-Out Data. Specifically, we compare the total number of retweets accumulated by users who do not perform intervention versus users who perform intervention to promote true news under the policy learned using different baselines. Let $I(\tau)$ refer to the the set of users who make interventions to their intensity for true news diffusion, in SED ($\tau \in [20, 30)$) and promote true news by time τ, according to the model, i.e., $I(\tau) = \{i|(\mathcal{N}_i(\tau, T, z) - \mathcal{N}_i(2K, T, z)) > 0\}$, and the remaining users who do not perform interventions, are referred to as $\neg I(\tau)$. We calculate the total number of users who retweeted the posts of users in $I(\tau)$ and $\neg I(\tau)$ between time $[\tau^{'}, \tau^{'} + \Delta)$ where $\tau^{'} = \tau + g$, and $g = \{0, 2, 5, 8\}$ indicates the gap or number of stages after which we want to measure the impact of users (in the future). We considered different values of $\Delta \in \{1, 2, 3, 4, 5\}$ and Table 7.1 reports the average. We see that the number of retweets accumulated by users who promote true news (I) is greater than that of users who do not perform interventions ($\neg I$) for our method DENPL by a large margin.

8 CONTRIBUTIONS

In this dissertation, we defined Social Reinforcement Learning (Chapter 3), for systems with a large number of agents with sparse interactions between them, in both fully observed and partially observed environments. We described the challenges of high-dimensionality and sparsity associated with Social RL problems, that makes it difficult to learn policies, especially in partially observable environments. We proposed ways to address these by utilizing properties of the social network structure, agent interactions and relations, by obtaining a compact model that better captures agent dependencies and is much more efficient to solve. We discussed several applications where the Social RL setting is applicable and characterized those with respect to the different attributes of the Social RL problem setting. In this chapter, we summarize the contributions of this dissertation, and outline avenues for future research.

8.1 Summary of Contributions

In Chapter 5, we proposed a centralized Social RL approach to capture the inter-agent dependencies and improve the sample efficiency per user for learning policies to incentivize users to spread more true news (to mitigate the impact of fake news). We designed a model to estimate the likely feedback for users based on both their network structure and the political bias of their peers. We demonstrated the gain in the performance of our method due to the user feedback modeled. Moreover, we effectively provided the user feedback as input to the policy function approximator, and thus, we avoided learning a separate complex model for each user (as in the standard reward shaping techniques). We also decoupled the post and response processes to approximate the joint action space more efficiently in order to reduce the computa-

tional complexity, as we dynamically optimized the intensity for the MHP governing the post events, and only learned parameters for the feedback events from historical data. We demonstrated that our proposed approach achieved better performance in terms of the reward and the number of distinct mitigated users. We showed the gain in the performance of our method due to appropriate selection of users and efficient allocation of incentive among them. We tested the efficacy of the users selected by our model and demonstrated that the selected users achieved greater number of retweets, indicating an increased true news spread.

In Chapter 6, we developed an efficient algorithm for centralized policy learning to overcome the challenges of high-dimensionality and sparsity, by clustering users dynamically to reduce the model size and achieve faster convergence as well. We aggregated the interactions of similar agents to overcome the problem of sparsity and curse of dimensionality, and thus, lowered the variance. We defined weighted centroids to ensure cluster alignment across different stages, while dynamically updating the cluster memberships, as required by the policy function approximator. We reduced the problem of learning policies for N users to that of learning policies for $\mathcal{C}$ clusters, and thus, the number of parameters to be learned from $\sim N^3$ to $\sim \mathcal{C}^3$ ($\mathcal{C} \ll N$). Thus, we lowered the computational complexity for centralized policy learning, while still considering all agent dependencies given large number of users. We designed latent features to cluster users based on their payoff and contribution to the goal, and thus, we provide a discriminative power to our model to differentiate between different types of users (and their activities), for efficient allocation of incentives among them under fixed budget. We also reduced the number of parameters by providing the latent features as input to the policy function via clusters, instead of adding them to the state representation. Thus, we better explore the action space, without increasing the state space, which also helps in faster convergence of our method. We showed that our dynamic clustering approach quickly learns better policy estimates, and outperformed different static clustering and non-clustering alternatives.

In Chapter 7, we created an effective extrapolated modeling approach for learning policies in partially observable environments where agents receive different local observations and rewards. We first learned the dependency between followees and followers, and extrapolated those to estimate hidden network state. We improvised the estimates of the hidden state and thus, the policy, by utilizing pairwise user interactions and peer-influence as attention. We developed a framework for sequential parameter sharing where a sequential update of parameters is performed by the users individually, based solely on their individual state, observations and rewards. We make our approach more privacy aware by only sharing parameters, and not the actual samples or trajectory information across users. We also provide more autonomy to the agents to learn their own actions and execute those in a decentralized fashion, rather than the system determining the actions for all users as in fully centralized learning. We showed that our approach is able to achieve performance comparable to the fully centralized learning method that assumes full observability and access to the complete network state, and also outperforms decentralized approaches that use baseline estimates of the hidden state. Thus, we illustrate the potential for Social RL methods to learn in a decentralized fashion, limiting the need for data sharing across agents.

We also developed new measures to improvise evaluation in scenarios where online application of policy is not feasible. Along with the reward measured from simulated data as in traditional approaches, we also measured the impact of users who modify their actions based on the learned policies using real-held out data. We designed the performance measure to emphasize more on the mitigation of distinct users over the mitigation of few users with high exposures.

8.2 Future Directions

There are a few directions that remain as future work and are very interesting to explore, for example:

In adversarial settings, agents have competing goals, and often do not have access to the state of other agents. The individual agent rewards can also be in conflict with the overall system objective, giving rise to the problem of *Social Dilemma* (e.g., [56, 176]). In Social RL setting, an example of social dilemma is when certain bots or users with malicious objectives want to intentionally spread fake news, which is in contrast to the system goal of promoting true news. In such scenarios, future work will involve exploring ways to model the *intentions* or *trustworthiness* of agents, and estimating the consequences of other agent's actions, and use those to improvise policy learning for an agent.

It might be possible to scale decentralized learning for large number of users by learning policies for individual agents concurrently. However, for this, it is critical to ensure that the policy learnt for an agent is responsive to the changes in the policies of other agents, i.e., the policies are *stationary*. When the agent policies are learnt independently without taking into account the actions of other agents, the stored samples used for learning an agent's policy can become obsolete as other agents update their policies concurrently resulting in non-stationary policies [102,177]. Thus, the policies that were learnt to be optimal in response to the past actions of other agents, can actually become worse as the other agents have already changed their policies meanwhile. Also, changes in the policy of an agent during learning can affect the optimal policy of other agents, leading to inaccurate estimates of the expected reward. Thus, in the future, it will be interesting to develop decentralized concurrent learning approaches for Social RL problems that yield stationary policies along with scaling to thousands of users.

We note that for reward calculations, we assume if user i follows user j, then all posts by j will be visible to i. However, for deployment in practice, systems would need to consider other factors like the time intervals when user j is online to determine if she is actually exposed to i's post. Future work can incorporate the probability of a user being online during certain time-intervals in a day and the probability of a user

i viewing user j's posts during those time-intervals, to improve reward measurements and overall performance.

REFERENCES

[1] Hunt Allcott and Matthew Gentzkow. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31(2):211–36, 2017.

[2] David MJ Lazer, Matthew A Baum, Yochai Benkler, Adam J Berinsky, Kelly M Greenhill, Filippo Menczer, Miriam J Metzger, Brendan Nyhan, Gordon Pennycook, David Rothschild, et al. The science of fake news. *Science*, 2018.

[3] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[4] Chao Yang, Robert Harkreader, and Guofei Gu. Empirical evaluation and new design for fighting evolving twitter spammers. *IEEE Transactions on Information Forensics and Security*, 8(8):1280–1293, 2013.

[5] Yang Liu and Yi-Fang Brook Wu. Early detection of fake news on social media through propagation path classification with recurrent and convolutional networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[6] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

[7] Kai Shu, H Russell Bernard, and Huan Liu. Studying fake news via network analysis: detection and mitigation. In *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*, pages 43–65. Springer, 2019.

[8] H.R.492. Biased algorithm deterrence act of 2019. *https://www.congress.gov/bill/116th-congress/house-bill/492/*, 2019.

[9] Gordon Pennycook and David G Rand. The implied truth effect: Attaching warnings to a subset of fake news stories increases perceived accuracy of stories without warnings. 2017.

[10] Norbert Schwarz, Lawrence J Sanna, Ian Skurnik, and Carolyn Yoon. Metacognitive experiences and the intricacies of setting people straight: Implications for debiasing and public information campaigns. *Advances in experimental social psychology*, 2007.

[11] Ullrich KH Ecker, Joshua L Hogan, and Stephan Lewandowsky. Reminders and repetition of misinformation: Helping or hindering its retraction? *Journal of Applied Research in Memory and Cognition*, 2017.

[12] Ceren Budak, Divyakant Agrawal, and Amr El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World wide web*. ACM, 2011.

[13] Xinran He, Guojie Song, Wei Chen, and Qingye Jiang. Influence blocking maximization in social networks under the competitive linear threshold model. In *Proceedings of the 2012 siam international conference on data mining*, pages 463–474. SIAM, 2012.

[14] Nam P Nguyen, Guanhua Yan, My T Thai, and Stephan Eidenbenz. Containment of misinformation spread in online social networks. In *Proceedings of the 4th Annual ACM Web Science Conference*, pages 213–222. ACM, 2012.

[15] Mehrdad Farajtabar, Jiachen Yang, Xiaojing Ye, Huan Xu, Rakshit Trivedi, Elias Khalil, Shuang Li, Le Song, and Hongyuan Zha. Fake news mitigation via point process based intervention. *arXiv preprint arXiv:1703.07823*, 2017.

[16] Craig Silverman. This analysis shows how viral fake election news stories outperformed real news on facebook. *BuzzFeed News*, 16, 2016.

[17] Dan M Kahan, Ellen Peters, Erica Cantrell Dawson, and Paul Slovic. Motivated numeracy and enlightened self-government. *Behavioural Public Policy*, 1(1):54–86, 2017.

[18] Rebecca M Jones, Leah H Somerville, Jian Li, Erika J Ruberry, Victoria Libby, Gary Glover, Henning U Voss, Douglas J Ballon, and BJ Casey. Behavioral and neural properties of social reinforcement learning. *Journal of Neuroscience*, 31(37):13039–13045, 2011.

[19] Yeoreum Lee and Youn-kyung Lim. Understanding the roles and influences of mediators from multiple social channels for health behavior change. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pages 1070–1079. ACM, 2015.

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[21] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[22] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18(5):620–634, 2010.

[23] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[24] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[25] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[26] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[27] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[28] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[30] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[31] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 167–176. International World Wide Web Conferences Steering Committee, 2018.

[32] Kleanthis Malialis, Sam Devlin, and Daniel Kudenko. Resource abstraction for reinforcement learning in multiagent congestion problems. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 503–511. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[33] Adrian K Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 980–987. IEEE Computer Society, 2004.

[34] Adrian K Agogino and Kagan Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[35] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 165–172. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[36] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[37] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[38] Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Le Song, and Hongyuan Zha. Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems*, pages 3247–3257, 2017.

[39] Manoel Horta Ribeiro, Pedro H Calais, Virgílio AF Almeida, and Wagner Meira Jr. "everything i disagree with is #fakenews": Correlating political polarization and spread of misinformation. *arXiv preprint arXiv:1706.05924*, 2017.

[40] Pedro Henrique Calais Guerra, Adriano Veloso, Wagner Meira Jr, and Virgílio Almeida. From bias to opinion: a transfer-learning approach to real-time sentiment analysis. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 150–158. ACM, 2011.

[41] Guillaume Deffuant, David Neau, Frederic Amblard, and Gérard Weisbuch. Mixing beliefs among interacting agents. *Advances in Complex Systems*, 3(01n04):87–98, 2000.

[42] Jan Lorenz. Continuous opinion dynamics under bounded confidence: A survey. *International Journal of Modern Physics C*, 18(12):1819–1838, 2007.

[43] Mahak Goindani and Jennifer Neville. Social reinforcement learning to combat fake news spread. *UAI*, 2019.

[44] Utkarsh Upadhyay, Abir De, and Manuel Gomez-Rodriguez. Deep reinforcement learning of marked temporal point processes. *arXiv preprint arXiv:1805.09360*, 2018.

[45] Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. Personalized ad recommendation systems for life-time value optimization with guarantees. In *IJCAI*, 2015.

[46] Sabina B Gesell, Kimberly D Bess, and Shari L Barkin. Understanding the social networks that form within the context of an obesity prevention intervention. *Journal of Obesity*, 2012, 2012.

[47] Katie Powell, John Wilcox, Angie Clonan, Paul Bissell, Louise Preston, Marian Peacock, and Michelle Holdsworth. The role of social networks in the development of overweight and obesity among adults: a scoping review. *BMC public health*, 15(1):996, 2015.

[48] Liuyan Shi, Liang Zhang, and Yun Lu. Evaluating social network-based weight loss interventions in chinese population: An agent-based simulation. *Plos one*, 15(8):e0236716, 2020.

[49] Jingwen Zhang, Devon Brackbill, Sijia Yang, and Damon Centola. Efficacy and causal mechanism of an online social media intervention to increase physical activity: results of a randomized controlled trial. *Preventive medicine reports*, 2:651–657, 2015.

[50] Damon Centola. Social media and the science of health behavior. *Circulation*, 127(21):2135–2144, 2013.

[51] Jun Wang, Weinan Zhang, Shuai Yuan, et al. Display advertising with real-time bidding (rtb) and behavioural targeting. *Foundations and Trends® in Information Retrieval*, 2017.

[52] He He, Jordan L. Boyd-Graber, Kevin Kwok, and Hal Daumé. Opponent modeling in deep reinforcement learning. In *ICML*, 2016.

[53] Anuj Mahajan and Theja Tulabandhula. Symmetry learning for function approximation in reinforcement learning. *arXiv preprint arXiv:1706.02999*, 2017.

[54] Martin Zinkevich and Tucker Balch. Symmetry in markov decision processes and its implications for single agent and multi agent learning. In *In Proceedings of the 18th International Conference on Machine Learning*. Citeseer, 2001.

[55] Patrick Mannion, Sam Devlin, Jim Duggan, and Enda Howley. Multi-agent credit assignment in stochastic resource management games. *The Knowledge Engineering Review*, 32, 2017.

[56] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[57] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438*, 2018.

[58] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2681–2690. JMLR. org, 2017.

[59] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[60] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[61] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate to solve riddles with deep distributed recurrent q-networks. *CoRR*, abs/1602.02672, 2016.

[62] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 1994.

[63] Marc Ponsen, Tom Croonenborghs, Karl Tuyls, Jan Ramon, Kurt Driessens, Jaap Van den Herik, and Eric Postma. Learning with whom to communicate using relational reinforcement learning. In *Interactive Collaborative Information Systems*. 2010.

[64] Tom Croonenborghs, Karl Tuyls, Jan Ramon, and Maurice Bruynooghe. Multi-agent relational reinforcement learning. In *International Workshop on Learning and Adaption in Multi-Agent Systems*, 2005.

[65] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine learning*, 2001.

[66] Nima Asgharbeygi, David Stracuzzi, and Pat Langley. Relational temporal difference learning. In *ICML*, 2006.

[67] Aditya Grover, Maruan Al-Shedivat, Jayesh K Gupta, Yuri Burda, and Harrison Edwards. Evaluating generalization in multiagent systems using agent-interaction graphs. In *AAMAS*, 2018.

[68] Aditya Grover, Maruan Al-Shedivat, Jayesh K Gupta, Yura Burda, and Harrison Edwards. Learning policy representations in multiagent systems. *arXiv preprint arXiv:1806.06464*, 2018.

[69] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. *arXiv preprint arXiv:1809.11044*, 2018.

[70] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[71] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[72] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent rl with global rewards. In *NeurIPS*, 2018.

[73] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Collective multiagent sequential decision making under uncertainty. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[74] Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *KDD*, 2018.

[75] Joel Z. Leibo, Julien Pérolat, Edward Hughes, Steven Wheelwright, Adam H. Marblestone, Edgar Du'enez-Guzm'an, Peter Sunehag, Iain Dunning, and Thore Graepel. Malthusian reinforcement learning. *CoRR*, abs/1812.07019, 2018.

[76] Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. Emergent coordination through competition. *CoRR*, abs/1902.07151, 2018.

[77] Aleksandra Malysheva, Tegg Taekyong Sung, Chae-Bong Sohn, Daniel Kudenko, and Aleksei Shpilman. Deep multi-agent reinforcement learning with relevance graphs. *arXiv preprint arXiv:1811.12557*, 2018.

[78] Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

[79] Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, and Chun-Yi Lee. A deep policy inference q-network for multi-agent systems. In *AAMAS*, 2018.

[80] Patrick Mannion, Karl Mason, Sam Devlin, Jim Duggan, and Enda Howley. Dynamic economic emissions dispatch optimisation using multi-agent reinforcement learning. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2016)*, 2016.

[81] Jakob N Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1811.01458*, 2018.

[82] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

[83] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P How. Learning to teach in cooperative multiagent reinforcement learning. *arXiv preprint arXiv:1805.07830*, 2018.

[84] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Michael Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. *arXiv preprint arXiv:1810.11187*, 2018.

[85] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[86] Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *AAMAS*, 2018.

[87] Dhouha Ben Noureddine, Atef Gharbi, and Samir Ben Ahmed. Multi-agent deep reinforcement learning for task allocation in dynamic environment. In *ICSOFT*, 2017.

[88] Thanh Le Chau Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Multi-agent deep reinforcement learning with human strategies. *CoRR*, abs/1806.04562, 2018.

[89] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Başar. Fully decentralized multi-agent reinforcement learning with networked agents. *arXiv preprint arXiv:1802.08757*, 2018.

[90] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. *CoRR*, abs/1612.07182, 2017.

[91] Patrick Mannion, Sam Devlin, Jim Duggan, and Enda Howley. Multi-agent credit assignment in stochastic resource management games. *Knowledge Eng. Review*, 32:e16, 2017.

[92] Saurabh Kumar, Pararth Shah, Dilek Z. Hakkani-Tür, and Larry P. Heck. Federated control with hierarchical multi-agent deep reinforcement learning. *CoRR*, abs/1712.08266, 2017.

[93] Mateusz Kurek and Wojciech Jaskowski. Heterogeneous team deep q-learning in low-dimensional multi-agent environments. *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016.

[94] Sherief Abdallah and Michael Kaisers. Addressing the policy-bias of q-learning by repeating updates. In *AAMAS*, 2013.

[95] Jeancarlo Arguello Calvo and Ivana Dusparic. Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In *AICS*, 2018.

[96] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[97] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multi-agent systems: A review of challenges, solutions and applications. *arXiv preprint arXiv:1812.11794*, 2018.

[98] Hoang M Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated multi-agent imitation learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1995–2003. JMLR. org, 2017.

[99] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

[100] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, pages 7254–7264, 2018.

[101] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

[102] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[103] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[104] Shihui Li, Yi Wu, Xinyue Cui, Hualing Dong, Fei Fang, and Stuart Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *AAAI 2019*, 2019.

[105] Matthew J. Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *CoRR*, abs/1511.04143, 2016.

[106] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *NIPS*, 2016.

[107] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. In *PloS one*, 2017.

[108] Logan Michael Yliniemi and Kagan Tumer. Multi-objective multiagent credit assignment in reinforcement learning and nsga-ii. *Soft Comput.*, 20:3869–3887, 2016.

[109] Adam Taylor, Ivana Dusparic, Edgar Galván López, Siobhán Clarke, and Vinny Cahill. Accelerating learning in multi-objective systems through transfer learning. *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2298–2305, 2014.

[110] Ivana Dusparic and Vinny Cahill. Distributed w-learning: Multi-policy optimization in self-organizing systems. *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 20–29, 2009.

[111] Kris Cao, Angeliki Lazaridou, Marc Lanctot, Joel Z Leibo, Karl Tuyls, and Stephen Clark. Emergent communication through negotiation. *arXiv preprint arXiv:1804.03980*, 2018.

[112] Trapit Bansal, Jakub W. Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *CoRR*, abs/1710.03748, 2018.

[113] Julien Pérolat, Joel Z. Leibo, Vinícius Flores Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *NIPS*, 2017.

[114] Edward Hughes, Joel Z. Leibo, Matthew Phillips, Karl Tuyls, Edgar A. Duéñez-Guzmán, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin R. McKee, Raphael Koster, Heather Roff, and Thore Graepel. Inequity aversion improves cooperation in intertemporal social dilemmas. In *NeurIPS*, 2018.

[115] Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2193–2201. ACM, 2018.

[116] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[117] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016.

[118] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. *arXiv preprint arXiv:1910.00091*, 2019.

[119] Karl Mason, Patrick Mannion, Jim Duggan, and Enda Howley. Applying multi-agent reinforcement learning to watershed management. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2016)*, 2016.

[120] Woojun Kim, Myungsik Cho, and Youngchul Sung. Message-dropout: An efficient training method for multi-agent deep reinforcement learning. *CoRR*, abs/1902.06527, 2019.

[121] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Çaglar Gülçehre, Pedro A. Ortega, Daniel Strouse, Joel Z. Leibo, and Nando de Freitas. Intrinsic social motivation via causal influence in multi-agent rl. *CoRR*, abs/1810.08647, 2018.

[122] Tianmin Shu and Yuandong Tian. Mˆ 3rl: Mind-aware multi-agent management reinforcement learning. *arXiv preprint arXiv:1810.00147*, 2018.

[123] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[124] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. *arXiv preprint arXiv:1802.09640*, 2018.

[125] Gregory Palmer, Rahul Savani, and Karl Tuyls. Negative update intervals in deep multi-agent reinforcement learning. *CoRR*, abs/1809.05096, 2018.

[126] Patrick Mannion, Jim Duggan, and Enda Howley. Learning traffic signal control with advice. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS 2015)*, 2015.

[127] Sandip Sen and Stéphane Airiau. Emergence of norms through social learning. In *IJCAI*, volume 1507, page 1512, 2007.

[128] Amit Prasad and Ivana Dusparic. Multi-agent deep reinforcement learning for zero energy communities. *CoRR*, abs/1810.03679, 2018.

[129] Kyrill Schmid, Lenz Belzner, Thomas Gabor, and Thomy Phan. Action markets in deep multi-agent reinforcement learning. In *International Conference on Artificial Neural Networks*, pages 240–249. Springer, 2018.

[130] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[131] Xiangyu Kong, Bo Xin, Fangchen Liu, and Yizhou Wang. Effective master-slave communication on a multi-agent deep reinforcement learning system. 2017.

[132] Xiangyu Kong, Bo Xin, Fangchen Liu, and Yizhou Wang. Revisiting the master-slave architecture in multi-agent deep reinforcement learning. *arXiv preprint arXiv:1712.07305*, 2017.

[133] Maximilian Hüttenrauch, Adrian Sosic, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *CoRR*, abs/1709.06011, 2017.

[134] Alvaro Ovalle Castaneda. Deep reinforcement learning variants of multi-agent learning algorithms. *Master's thesis, School of Informatics, University of Edinburgh*, 2016.

[135] Ozsel Kilinc and Giovanni Montana. Multi-agent deep reinforcement learning with extremely noisy observations. *arXiv preprint arXiv:1812.00922*, 2018.

[136] Ashish Goel, Kamesh Munagala, Aneesh Sharma, and Hongyang Zhang. A note on modeling retweet cascades on twitter. In *International Workshop on Algorithms and Models for the Web-Graph*. Springer, 2015.

[137] Dar Meshi, Diana I Tamir, and Hauke R Heekeren. The emerging neuroscience of social media. *Trends in cognitive sciences*, 19(12):771–782, 2015.

[138] Eveline A Crone and Elly A Konijn. Media use and brain development during adolescence. *Nature communications*, 9(1):588, 2018.

[139] Yu Wang, Jiebo Luo, Richard Niemi, Yuncheng Li, and Tianran Hu. Catching fire via" likes": Inferring topic preferences of trump followers on twitter. In *ICWSM*, pages 719–722, 2016.

[140] Duanbing Chen, Linyuan Lü, Ming-Sheng Shang, Yi-Cheng Zhang, and Tao Zhou. Identifying influential nodes in complex networks. *Physica a: Statistical mechanics and its applications*, 391(4):1777–1787, 2012.

[141] Yosihiko Ogata. On lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1):23–31, 1981.

[142] PA W Lewis and Gerald S Shedler. Simulation of nonhomogeneous poisson processes by thinning. *Naval research logistics quarterly*, 26(3):403–413, 1979.

[143] Mehrdad Farajtabar, Xiaojing Ye, Sahar Harati, Le Song, and Hongyuan Zha. Multistage campaigning in social networks. In *Advances in Neural Information Processing Systems*, pages 4718–4726, 2016.

[144] Aleksandr Simma and Michael I Jordan. Modeling events with cascades of poisson processes. *arXiv preprint arXiv:1203.3516*, 2012.

[145] Michela Del Vicario, Antonio Scala, Guido Caldarelli, H Eugene Stanley, and Walter Quattrociocchi. Modeling confirmation bias and polarization. *Scientific reports*, 7:40391, 2017.

[146] Jing Ma, Wei Gao, and Kam-Fai Wong. Detect rumors in microblog posts using propagation structure via kernel learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 708–717, 2017.

[147] Ke Zhou, Hongyuan Zha, and Le Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Artificial Intelligence and Statistics*, pages 641–649, 2013.

[148] Mehrdad Farajtabar, Safoora Yousefi, Long Q Tran, Le Song, and Hongyuan Zha. A continuous-time mutually-exciting point process framework for prioritizing events in social media. *arXiv preprint arXiv:1511.04145*, 2015.

[149] Matthew Engelhard, Hongteng Xu, Lawrence Carin, Jason A Oliver, Matthew Hallyburton, and F Joseph McClernon. Predicting smoking events with a time-varying semi-parametric hawkes process model. *arXiv preprint arXiv:1809.01740*, 2018.

[150] E. Bacry, M. Bompaire, S. Gaïffas, and S. Poulsen. tick: a Python library for statistical learning, with a particular emphasis on time-dependent modeling. *ArXiv e-prints*, July 2017.

[151] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[152] Ankur P Parikh, Asela Gunawardana, and Chris Meek. Conjoint modeling of temporal dependencies in event streams. 2012.

[153] Zhen Qin and Christian R Shelton. Auxiliary gibbs sampling for inference in piecewise-constant conditional intensity models. In *UAI*, pages 722–731, 2015.

[154] Mehrdad Farajtabar, Yichen Wang, Manuel Gomez Rodriguez, Shuang Li, Hongyuan Zha, and Le Song. Coevolve: A joint point process model for information diffusion and network co-evolution. In *Advances in Neural Information Processing Systems*, pages 1954–1962, 2015.

[155] Kai Shu, Suhang Wang, and Huan Liu. Understanding user profiles on social media for fake news detection. In *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pages 430–435. IEEE, 2018.

[156] Michael Mccord and M Chuah. Spam detection on twitter using traditional classifiers. In *international conference on Autonomic and trusted computing*, pages 175–186. Springer, 2011.

[157] Jianan Yue. *Analyzing and Detecting Social Spammers with Robust Features*. PhD thesis, McGill University Libraries, 2017.

[158] Alexandre Bovet and Hernán A Makse. Influence of fake news in twitter during the 2016 us presidential election. *Nature communications*, 10(1):7, 2019.

[159] Lion Gu, Vladimir Kropotov, and Fyodor Yarochkin. The fake news machine: how propagandists abuse the internet and manipulate the public. *Trend Micro*, 2017.

[160] Nor Athiyah Abdullah, Dai Nishioka, Yuko Tanaka, and Yuko Murayama. Why i retweet? exploring user's perspective on decision-making of information spreading during disasters. In *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

[161] Vincenza Carchiolo, Alessandro Longheu, Michele Malgeri, Giuseppe Mangioni, and M Previti. Terrorism and war: Twitter cascade analysis. In *International Symposium on Intelligent and Distributed Computing*. Springer, 2018.

[162] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*. AcM, 2010.

[163] Thi Bich Ngoc Hoang and Josiane Mothe. Predicting information diffusion on twitter–analysis of predictive features. *Journal of computational science*, 28:257–264, 2018.

[164] Feiyun Zhu, Jun Guo, Zheng Xu, Peng Liao, Liu Yang, and Junzhou Huang. Group-driven reinforcement learning for personalized mhealth intervention. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 590–598. Springer, 2018.

[165] Ali el Hassouni, Mark Hoogendoorn, Martijn van Otterlo, and Eduardo Barbaro. Personalization of health interventions using cluster-based reinforcement learning. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 467–475. Springer, 2018.

[166] Rémi Lemonnier, Kevin Scaman, and Argyris Kalogeratos. Multivariate hawkes processes for large-scale inference. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[167] Nazanin Alipourfard, Buddhika Nettasinghe, Andrés Abeliuk, Vikram Krishnamurthy, and Kristina Lerman. Friendship paradox biases perceptions in directed networks. *Nature communications*, 11(1):1–9, 2020.

[168] Eun Lee, Fariba Karimi, Claudia Wagner, Hang-Hyun Jo, Markus Strohmaier, and Mirta Galesic. Homophily and minority-group size explain perception biases in social networks. *Nature human behaviour*, 3(10):1078–1087, 2019.

[169] Mahak Goindani and Jennifer Neville. Cluster-based social reinforcement learning. *International Conference on Autonomous Agents and Multi-Agent Systems*, 2020.

[170] Mahak Goindani and Jennifer Neville. Social reinforcement learning. In *2020 AAAI Spring Symposium Series*, 2020.

[171] Young Joon Park, Yoon Sang Cho, and Seoung Bum Kim. Multi-agent reinforcement learning with approximate model learning for competitive games. *PloS one*, 14(9), 2019.

[172] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[173] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[174] Guifang Liu, Huaiqian Bao, and Baokun Han. A stacked autoencoder-based deep neural network for achieving gearbox fault diagnosis. *Mathematical Problems in Engineering*, 2018, 2018.

[175] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[176] Alexander Peysakhovich and Adam Lerer. Towards ai that can solve social dilemmas. In *2018 AAAI SSS*, 2018.

[177] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1146–1155. JMLR. org, 2017.