Summer 2020

Empirical Studies of Deep Learning on Information Diffusion on Social Networks and Collective Task Learning for Swarm Robotics

Trung T. Nguyen
University of Southern Mississippi

EMPIRICAL STUDIES OF DEEP LEARNING ON INFORMATION DIFFUSION ON SOCIAL NETWORKS AND COLLECTIVE TASK LEARNING FOR SWARM ROBOTICS

by

Trung T. Nguyen

A Dissertation
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Computing Sciences and Computer Engineering
at The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved by:

Dr. Andrew H. Sung, Committee Chair
Dr. Bikramjit Banerjee
Dr. Ramakalavathi Marapareddy
Dr. Parthapratim Biswas
Dr. Sungwook Lee

August 2020

Published by the Graduate School

ABSTRACT

Researchers in multiple disciplines have recently adopted deep learning because of its ability of high accuracy representation learning from big and complex data. My research goal in this thesis is developing deep learning models for information diffusion analysis on social networks and collective tasks learning in swarm robotics.

Firstly, the information diffusion on social networks is modeled as a multivariate time series in three dimensions with ten features. Then, we applied time-series clustering algorithms with Dynamic Time Warping to discover different patterns of our models. Then, we build a prediction model based on LSTM, which outperforms traditional time-series prediction methods. Extending the first study, we conduct the second study to measure the users' influence on social networks. Our deep learning model outperforms the baseline Linear Influence Model in modeling the global influences of nodes over time and predicting the temporal volume of information diffusion processes. Our third study proposes the Multi-Feed Weighted Topic Embeddings (MFWTE) model to analyze user network interactions and topic diffusion patterns on Twitter. Our model is evaluated on the friendship recommendations and retweet link prediction tasks. The performance shows that our MFWTE model outperforms various single feed methods and can help to study topic diffusion patterns.

In swarm robotics, we employ deep reinforcement learning methods to solve collective task learning problems. The fourth study proposes a cumulative training method using transfer learning to develop a multi-robot collision avoidance navigation system. Our approach improves the shared policy between multiple agents through transfer learning, reward shaping, and multi-stage training. The performance shows that our method helps to

produce a robust navigation plan that generalizes well to complex indoor scenarios. Finally, we propose a new multi-robot foraging approach based on Reinforcement learning as a Rehearsal framework. This approach takes both the visible and hidden features, trains through Deep Q-Learning, and generates a robust policy to decide the role choice between walker and beacon. While the hidden features are available during training time, we make them available during execution time by learning a generation model based on Mixture Density Networks through deep learning.

ACKNOWLEDGMENTS

There are many people I would like to thank for stimulating, challenging, and encouraging me to complete this work.

Firstly, I would like to thank my primary supervisor, Dr. Andrew H. Sung, for his guidance and support. During the last four years, he has invested plenty of time and effort to guide and encourage me on both scientific research, teaching, and writing skills.

Secondly, I would like to thank my co-supervisor, Dr. Bikramjit Banerjee, for his inspiration and guidance, especially in the field of reinforcement learning. We have recently collaborated on researching of learning-based approach for swarm robotics tasks.

I also thank other committee members, including Dr. Parthapratim Biswas, Dr. Sungwook Lee, and Dr. Ramakalavathi Marapareddy, for providing valuable feedback and suggestions during the time I worked on this thesis.

I would also like to thank Amartya Hatua and Pujan Paudel, who are also the collaborators in many of my research work. We have shared a lot of great conversations and wonderful time. Next, I also want to thank Asheshbabu Pothuraju and other master students for helping me conducting some data collection, preprocessing, and experiments.

Thank you to the School of Computing Sciences and Computing Engineering department for supporting me during the last four years and providing a supportive community. A part of this work was supported by a National Science Foundation grant IIS-152681. Also, there are some colleagues who I was fortunate to learn, collaborate, and share knowledge & experience in other research/academic work: (in alphabetical order) Charan Gudla, Gabriel Idakwo, Md Rana, Sarbagya Shakya, and Haipeng Tang.

DEDICATION

This thesis is dedicated to my wife, Ha, my beloved children, and my parents. There are no words good enough to express how I feel. Your encouragement and kind support towards my academic works have been tremendous.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF SCHEMES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
CART	Classification and Regression Trees
CNN	Convolution Neural Network
CVI	Cluster Validity Index
DNN	Deep Neural Network
DQN	Deep Q-Network
DTW	Dynamic Time Warping
GPU	Graphical Processing Units
LDA	Latent Dirichlet Allocation
LIM	Linear Influence Model
LSTM	Long Short Term Memory
MDP	Markov Decision Process
MFWTE	Multi-Feed Weighted Topic Embeddings
MLP	Multi-layer Perceptron
MSE	Mean Squared Error
ORCA	Optimal Reciprocal Collision Avoidance
OSN	Online Social Network
PPO	Proximal Policy Optimization
POMDP	Partially Observable Markov Decision Process
RBF	Radial Basis Function

RLaR	Reinforcement Learning as a Rehearsal
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RVO	Reciprocal Velocity Obstacles
SVM	Support Vector Machines
TDNN	Time Delay Neural Network
TRPO	Trust Region Policy Optimization

CHAPTER I - INTRODUCTION

1.1 Motivation

Recently, machine learning, especially deep learning, has been widely used in different disciplines. Deep learning is a new subfield of machine learning that takes advantage of the availability of big data, the automatic feature engineering capabilities, to achieve higher performance in multiple applications when comparing to classical machine learning approaches. My research goal in this thesis is to develop deep learning models for information diffusion analysis on social networks and collective tasks learning in swarm robotics. There are some reasons that deep learning is well fit the two selected domains. While studying information diffusion on social networks, we have to deal with the enormous data volume and problem complexity because of the vast and complex user network interactions. Besides, in the domain of swarm robotics, there are two major challenges. The first one is the problem of curse dimension (the problem's dimensionality grows exponentially with the increasing number of robots), and the second one is the partially observable state problem (locally sensed information rather than the global observation state). That is the reason why deep reinforcement learning algorithms can be used to accelerate the self-learning capabilities of swarm agents to perform multi-tasking.

1.1.1 Information Diffusion on Social Networks

Nowadays, social networks have become one of the most critical communication channels for viral marketing, political or promotion campaigns, opinion formation, and others. Social networks provide users the platform to connect and contribute to the information propaganda (or information diffusion) by sharing and spreading information.

Learning the behaviors/patterns of such processes can help us to understand and predict different characteristics (especially public opinions, popularity, etc.) of numerous events in real life. Since there is increasing adoption of social media by individuals and organizations, information diffusion analysis on social networks becomes a growing interest research topic. In this case, information diffusion analysis means modeling, quantifying, and predicting the characteristics and impact of information flows on social media platforms.

1.1.2 Collective Tasks Learning in Swarm Robotics

Recently, swarm robotics have been widely used in many areas such as precision farming, targeted material delivery, video surveillance, cyber defense, and manufacturing, etc. Swarm robotics has some good qualities, such as robustness, flexibility, and scalability (Şahin, 2004; Brambilla, Ferrante, Birattari, & Dorigo, 2013). These make swarm robotics fit for those above problems. Swarm robotics can be considered as a combination of a multi-robots system with swarm-intelligent capabilities, which simulate the social structures and interactions in the system (Tan & Zheng, 2013).

The robots in a swarm are designed to be autonomous and decentralized. These robots have physical bodies, interact with surrounding worlds through sensors, actuators, etc., and can operate on their own. Robots in swarm often have limited local sensing, communication, and computing/memory capabilities. Therefore, for scalability, swarm robotics contain a large enough number of individual robots. Next, for the flexibility property, the swarm should consist of a few homogeneous groups of robots. The major of individual robots should have similar design, structure, and capabilities for mass production and flexibility. An essential feature of swarm robotics is its performance comes

from the *collective behavior* of a whole group, rather than from the outstanding performance of an individual. This cooperative behavior is an important research topic in swarm robotics. In this work, it is one of the main targets of my research, in which the focus is on the learning-based approaches.

In swarm robotics, navigation is a fundamental task. In this context, each robot is assigned a destination location to navigate. During the navigation period, each robot may collide with various types of obstacles (either stationary or moving entities such as other robots). The biggest challenge here is developing a safe and effective collision avoidance multi-robot navigation policy. Each robot has a set of auxiliary devices, such as proximity sensors, actuators, and cameras, etc., to provide the surrounding observations. Efficient navigation has to generate the steering commands from those observations from those devices with the lowest probability that such collisions can happen.

Besides navigation, there are numerous tasks such as grasping an object, transporting, communication, etc. can be performed by swarm agents that require coordination and cooperation actions. Multi-robot foraging is an exemplary example of such a type of problem that has a lot of promising applications in real life.

In this work, deep reinforcement learning approaches are introduced to solve the multi-robot navigation and multi-robot foraging in swarm robotics.

1.2 Contributions

This section describes the contributions of this work in applying deep learning for studying information diffusion on social networks and collective task learning in swarm robotics. The contributions are listed as the following:

1. Traditional information diffusion modeling methods on social networks depend on explicit knowledge of the system, which is not easy to capture because there are many complicated components to track, such as topics/memes, tags, URLs, mentioned users, etc. Therefore, a multivariate time series problem in three dimensions with ten features is used to model the information diffusion (Hatua, Nguyen, & Sung, 2017): A) volume: #tweet, #retweet, b) Influence: #direct and #indirect, c) Sentiment: percentage of negative, neutral, positive, and the average score of negative, neutral, positive. After the previous step of information diffusion modeling using multivariate time series with ten features, we conducted a study to analyze several patterns that exist in the dataset. Time series clustering algorithms are utilized to discover such patterns. We applied clustering algorithms such as partitional clustering, hierarchical clustering, TADPole clustering with Dynamic Time Warping (DTW) (Friedman, Hastie, & Tibshirani, 2001) distance to determine different information diffusion patterns and their diffusion rate. Based on the above step, we discovered different clusters corresponding to the ten features in our model. Also, we used k-NN with DTW distance to detect the information propaganda patterns of a new hashtag (classify it into the corresponding cluster).

The final step is constructing a time series predicting model to forecast the information diffusion characteristics in our models. The performance comparison shows that our deep learning prediction models based on Long-Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) outperform the traditional time series forecasting methods (ARIMA) (Box et al. 2015) for such prediction task.

2. We propose a model to quantify the influence of each user based on their Tweet behavior (Nguyen, Hatua, Pothuraju, & Sung, 2018). Our proposed model, motivated by

the Linear Influence Model (LIM) (Yang and Leskovec, 2010), is a nonparametric model, which does not depend on the structure of the underline network. Instead of using the least square method to measure the influence of users in the system, deep learning models are used in our approaches. Various learning machines, such as Deep Neural Network (DNN) and Long Short-Term Memory (LSTM), are used in experiments to study the diffusion rate of nodes over time and build the prediction models. In this research, we will use LIM as the baseline model for performance comparison. We propose to use neural network models such as DNN or LSTM to capture the user influence time series and develop the global diffusion rate prediction system using the regression model at the final output layer. The performance evaluation shows that our proposed influence model outperforms the baseline Linear Influence Model with higher accuracy when modeling the global influences of nodes over time and predicting the temporal volume of information diffusion processes.

3. A good understanding of user network interaction behavior on social networks can be used for real-world applications like political campaigns, viral marketing, targeted advertising, etc. Therefore, a Multi-Feed Weighted Topic Embeddings (MFWTE) (Paudel, Hatua, Nguyen, & Sung, 2019) model was proposed to analyze topic diffusion patterns and user network interaction on Twitter. The first part of our proposed methodology is extracting weighted topic embeddings from several views of users' feed and based on their tweet/retweet behaviors. There are four user views: the authored tweets, replied tweets, retweeted tweets, and favorited tweets. Later, we compare the performance of our proposed method with two other topic modeling algorithms: a) Latent Dirichlet Allocation and b) Twitter-Latent Dirichlet Allocation on the retweet link prediction friendship and recommendation tasks. The users are also divided into multiple tiers based on their activity

behaviors regarding particular topics, and the effectiveness of MFWTE is evaluated using these settings. The performance demonstrates the effectiveness of our MFWTE model through the evaluation of the retweet behavior prediction and friendship recommendation tasks and discovering information diffusion patterns through topic models.

4. We proposed a cumulative training approach based on deep reinforcement learning algorithms to develop an efficient multi-robot collision avoidance navigation problem (Nguyen, Hatua, & Sung, 2019). Our proposed method helps to build a generalized and robust policy for such a problem in complex environments that include short-range to long-range navigation tasks. The core of the learning method consists of two modified policy gradient algorithm (Proximal Policy Optimization (PPO) (Schulman, Wolski, et al., 2017) and Trust Region Policy Optimization (TRPO) (Schulman et al., 2015)). Our approach does not require direct communication between agents but only a communication channel with a master to receive the navigation tasks and the learning feedbacks during the training phase. Our proposed approach shows better performance compared to previous methods, reduces training time, and has better generalization performance in complicated indoor settings.

5. A deep reinforcement learning approach, which is developed from the Reinforcement learning as a rehearsal (RLaR) (Kraemer & Banerjee, 2016) framework, is proposed to solve the multi-robot foraging problem. The visible and hidden features have been designed for this application. Deep Q-Learning (DQN) (Gu, Lillicrap, Sutskever, & Levine, 2016) policy was trained to replace the heuristic decision (Hoff, Sagoff, Wood, & Nagpal, 2010) for optimizing the role selection between walker and beacon (Nguyen & Banerjee, 2020). The hidden features are available during training time, but not available

in executing time. Dealing with the above problem, we collected the hidden features during the experiments. We later used deep learning to train the Mixture Density Networks (MDN) (Bishop, 1994) model to generate these hidden features from the visible features. The performance shows that our approach can substantially improve the foraging metric (the number of food returns over time). Moreover, our approach can be generalized to other partially observable problems as well as other multi-robot learning problems.

1.3 Thesis structure

This thesis is constructed as follows. First, chapter 1 outlines specific research objectives and associated approaches to the proposed investigation. Next, chapter 2-4 continues with background information and reviews relevant research studies illustrating machine learning and deep learning background (chapter 2), information diffusion on social networks (chapter 3), and collective task learning in swarm robotics (chapter 4). Chapter 5 introduces different deep learning approaches to perform information diffusion studies in social networks. Next, chapter 6 follows with the research design and methods for the remaining studies on collective task learning of swarm robotics. Finally, chapter 7 concludes with summarizing my research objectives, the works I have done, and discuss the future works.

CHAPTER II – MACHINE LEARNING & DEEP LEARNING BACKGROUND

Most traditional machine learning algorithms can work well because of the handcrafted features from inputs by humans. In this case, machine learning algorithms will generally be the processes of optimizing the weights to obtain the best prediction model of outputs from inputs. Deep learning becomes popular because of its capabilities of representation learning. Through multiple levels of complexity and abstraction, deep learning can learn useful features and finally helps to build a better final prediction model at the end.

This chapter starts with the review of machine learning and deep learning, the reason why deep learning became popular recently, and the basics of the neural network. Later advanced deep learning architectures such as Convolution Neural Network (CNN), Recurrent Neural Network (RNN), and finally, Reinforcement Learning and Deep Reinforcement Learning are discussed.

2.1 Introduction to Machine Learning and Deep Learning

According to (Russell & Norvig, 2009), a learning agent will improve its performance on a specific task by receiving observations from its surrounding world. Machine learning is a technique that can help agents to learn, and in this case, it tries to learn from data, a dictionary of input-output pairs, a function that predicts the outputs from the inputs. Machine learning has recently become popular because of its tremendous applications for massive data processing and artificial intelligence (AI). Previously, humans could grasp and analyze data based on domain experts and accumulated knowledge throughout their working experiences. However, when data becomes enormous and more complicated with many variables and features, machine learning and especially deep

learning has emerged to be the best method to handle massive datasets nowadays. Machine learning relies much on statistical foundations, but it is not purely statistics but also programming and mathematics. They can help to find patterns in data, generalizing rules to data, and using those abilities to predict future data (Wilkins, 2019). Deep Learning (LeCun, Bengio, & Hinton, 2015) is a subset of machine learning techniques to use multi-layer architecture for representation learning, in which each layer will transform the data into a more abstract level of representation.

Supervised learning, unsupervised learning, and reinforcement learning are three classes of machine learning techniques. Supervised learning is a class of machine learning approaches that use the labeled data to learn the input features to classify/predict the unlabeled data. If the labels of the output are categories, such as spam/not spam, this kind of prediction task is called classification. Besides, the prediction task is called regression if the output labels are from a continuous domain, such as the stock price. The underlying principle of supervised learning is to learn a function f that maps from input features to output labels (either categorical or continuous) over the collected dataset. For example, we can use machine learning to develop a spam mail classification engine. From the dataset of spam and not spam emails, we extract different features from such datasets, such as the frequency of specific terms (words), sender information, email address, IP, etc. Later, we can apply different machine learning techniques such as Decision Tree (Quinlan, 1983), Random Forest (Ho T. K., 1995), Support Vector Machines (SVM) (Ben-Hur, Horn, Siegelmann, & Vapnik, 2001), Artificial Neural Network (ANN), etc. for training a spam / not spam mail classification model.

Unsupervised learning is another machine learning class that learns the patterns of the input data without explicit labels of output. Clustering is usually used in unsupervised learning techniques, where input features are classified into potential useful clusters based on the high similarity between samples in the same group and low similarity between samples from different clusters. K-means (MacQueen, 1967) clustering algorithm is a popular cluster analysis technique. It aims to separate n data samples into k clusters so that the Euclidean distances between input data samples in each group are minimized. Unsupervised learning has been broadly applied in different areas such as market segmentation, credit card fraud detection in banking, object segmentation in an image, gene clustering, content-based document classification, etc. Furthermore, in a recommendation system, the clustering technique is used to group users who have similar profiles or buying/consuming items habits. Unsupervised learning can also help to discover useful association rules, which demonstrates the high probability of co-occurrence of items in a data collection.

Reinforcement Learning is a particular machine learning class that helps an artificial agent to learn some desired behaviors in a designed environment. In reinforcement learning, the training data does not exist at the beginning. The agent repeatedly interacts with the environment (perform different actions), collects the new experiences data (the new state and feedback of such environment), and use them to adjust its action policy. This mechanism is very similar to the way humans and animals learn new skills or knowledge. Here, we need to deal with a sequential decision problem in which the cumulative reward is maximized through time. Recent advances in reinforcement learning and deep learning have led to the creation of Google AlphaGo Zero, with the capabilities of mastering the Go

game in short training time through self-play, that win over the world champion human player in such game (Silver et al., 2017). Therefore, deep reinforcement learning, a combination of representation learning capability from deep learning and efficient decision-making learning capability from reinforcement learning, has become the cutting-edge research paradigm that can lead to general artificial intelligence. Reinforcement learning is very suitable for decision-making problem that depends on short-term or long-term reward such as robotics, computer games, multi-agent systems, vehicular navigation, logistics, networking, etc.

Artificial neural networks (ANN) have a long history because its first computational model, named perceptron, was first introduced in the 1950s (Rosenblatt, 1958). At that time, because of the limited computing power, ANN existed on research papers only rather than from a business perspective. Even, people almost forgot about ANN during the AI research winter in the 1980s and 1990s. However, ANN revives in the 2000s because of the improving computing capability and availability of massive data volumes (Wilkins, 2019). ANN or shallow neural networks has two limitations: computing power to simulate layers of artificial neurons, and the requirements of enough data and feature selection. Deep learning is the successor of ANN by merely creating more layers of depth compared to traditional ANNs. The introduction of powerful graphics processing units (GPU) has increased the computational capabilities of our systems. As GPUs are fast and better year after year, many deep learning projects have adopted them to improve their training performance. The main power of deep learning is the exceptional ability of representation learning of input features. The features are abstracted through different levels of non-linear hidden layers of deep neural networks. Therefore, deep learning helps

us to overcome the problem of feature engineering that traditional machine learning techniques often suffer. Nowadays, a lot of successful applications of deep learning have been introduced, such as computer vision, speech recognition, language processing, deep reinforcement learning, and self-driving car, etc.

2.2 Different Machine Learning Algorithms

This subsection will introduce some of the most basics and popular techniques in machine learning.

2.2.1 Linear and logistic regression

In machine learning, regression is a method to model a target-dependent variable based on independent variables. This method is used to forecast or find the causal and effect between variables. For example, a group of features of a house such as size, number of bathrooms, number of floors, and location, is given. We try to predict the price of such a house. Linear regression and logistic regression are two fundamental and popular methods in regression. While linear regression algorithms are mostly used for prediction tasks, logistic regression is used for classification.

Linear regression algorithms try to find a linear dependence between the dependent variable (y) on the independent variables set ($X = \{x_0, x_1, \dots, x_n\}$). Here, we can use $\vec{x}$ as the input features vector to represents the values of such set X. The red line in Figure 2.1 displays the best line to fit the given data samples. In this case, a straight line can represent the linear relationship between the two variables on the vertical and horizontal axis. The linear regression model can be described using the following equation:

$$\hat{y} = \vec{w}.\vec{x} + \vec{b} = \sum_i (w_i x_i + b_i)$$

where $\vec{w}$ denotes the weight vector and $\vec{b}$ indicates the bias vector. Linear regression algorithms will try to find the optimal value of $\vec{w}$ and $\vec{b}$ so that the values of $\hat{y}$ on every inputting datapoint are close to the actual values of y. In other terms, a *cost* function, which defines the differences between the predicted values and actual values, can be optimized to find such parameters. The *mean square error* (*MSE*) is a popular cost function, which is characterized as the next equation:

$$argmin\frac{1}{n}\sum_{i=1}^{n}(\hat{y}-y)^2 = argmin\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{i}(\vec{w}\vec{x}+\vec{b})-y\right)^2$$

To optimize the above cost function to obtain the optimum value of $\vec{w}$ and $\vec{b}$, gradient descent (Hao & Lesnic, 2000; Ruder, 2016) method is usually used. In this method, at each updating step, the derivatives of the cost function on each parameter w_i or b_i need to be calculated. These derivatives tell us the way to update the current values of w_i or b_i to lead our cost function to the minimal value.

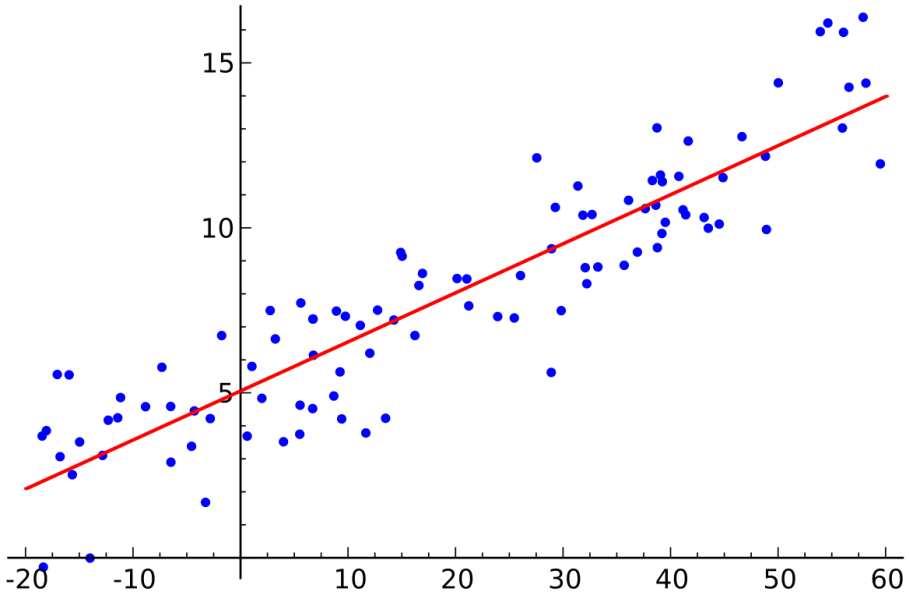


Figure 2.1 *Example of Linear Regression.*

Logistic regression: Compared to linear regression, rather than outputting the value $z = \vec{w}\vec{x} + \vec{b}$, logistic regression applies another particular function $\sigma(z)$ to squash the value into the range of $[0,1]$. One of such popular function is sigmoid, which is specified by the equation $s(z) = \frac{1}{1+e^{-z}}$. That is the reason why logistic regression can be used for classification tasks because its output can be considered as a probability. In this case, the closer of its output to 1, the more likely the expected hypothesis is. For example, logistic regression can be used to model the probability to classify a mail is spam or not. In this case, the higher the output value, the more confident of a mail sample is a spam one. For the classification task, a decision rule based on threshold value can be introduced to transform the logistic regression output into relevant classes. For example, if $\sigma(z) \geq 0.5$, a mail will be classified as spam and vice versa. The training process of logistic regression is very similar to the training of linear regression, but there is an extra learning step to squash the similar output of linear regression into the $[0,1]$ interval.

2.2.2 Decision Tree and Random Forests

Decision Tree. It is a member of the tree-based learning algorithms family that is one of the best and broadly applied machine learning methods. Such popular algorithms are the Classification & Regression Trees (CART) (Breiman, Friedman, Stone, & Olshen, 1984) methodology, the Iterative Dichotomiser 3 (ID3) (Quinlan J. R., 1986), C4.5 (Quinlan J. R., 2014), etc. These algorithms can be adapted to any classification or regression problems with high accuracy, stability, and interpretability. Compared to the linear models introduced in the previous section, these algorithms can work well on the non-linear relationships of input and output variables. A flowchart-like tree structure can represent a decision tree. Here, each node denotes a test condition on a variable, then

branches into multiple children nodes, and continues until finally reaching the leaf nodes. In the classification problem, the leaf node in the decision tree model represents the class labels. However, in the regression problem, it represents the output value (of a continuous variable or a range of value to be sampled by a distribution, i.e., Gaussian). A path from the root node to the leaf node can be understood as the combination of all computations over the nodes to have the classification rules or regression rules. Figure 2.2 displays an illustration of a decision tree example.

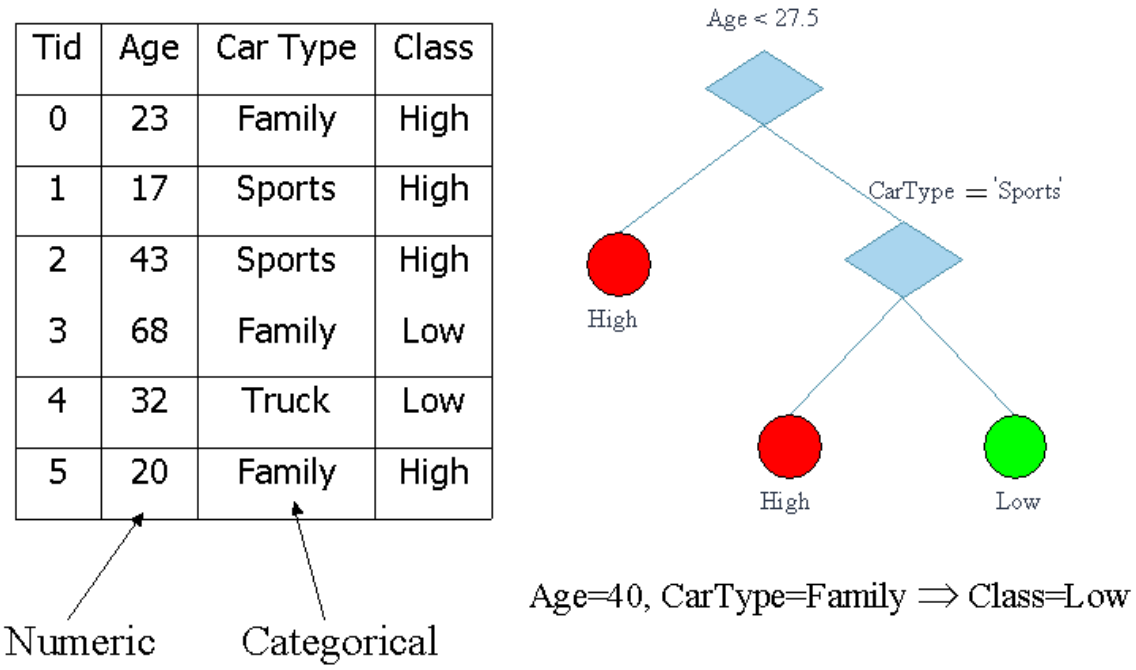


Figure 2.2 *A visualization of a decision tree example (right) for the dataset (left)*

Random Forests: An ensemble learning method that combines several decision trees to produce a better classification/prediction model. In 1995, the random decision forests (Ho T. K., 1995) algorithm was first introduced. Breiman extended it in 2001 and coined the named Random Forests (Breiman, 2001). The training algorithm of random forests uses the idea of bagging, or bootstrapping aggregating, on the multiple decision tree

learners. This process repeatedly applies the random sampling with replacement technique on the training dataset while fitting the decision trees. For the classification task, the majority vote of all decision trees in the final random forest model is used to determine the prediction class (Liaw & Wiener, 2002). Besides, in the regression task, the prediction output is determined by the averaging of all individual regression trees in the final model. Although the high computation and complexity of random forests, this algorithm is well-known for generalization by preventing the overfitting problem and reducing the variance.

2.2.3 Support Vector Machine (SVM)

This one is a popular supervised learning approach. Vapnik and Chervonenkis proposed the original idea of the maximum-margin hyperplane in 1963. SVM became popular from 1992 when the kernel trick was applied to the maximum-margin hyperplane algorithm for the non-linear classifier (Cortes & Vapnik, 1995). Figure 2.3 presents the example of two SVM classifiers on the iris flower dataset[1] to classify the samples between three classes: Setosa, Versicolor, and Virginia. SVM tries to find the best largest-margin hyperplane, which separated the samples of two classes. Given a training data set that belongs to two classes, each data point is presented by a feature vector of length m. The classification problem of these two classes is called linear if we can find a $m-1$ dimensional hyperplane to separate the data points of two classes. SVM looks for the maximum-margin hyperplane, which has the maximal distance to the nearest data points from each category. However, there are a lot of problems that are not linearly separable. That is where the kernel trick proposed by Vapnik is best used for, to transform the original problem into higher-dimensional space. It is supposed that SVM can find the best separator

[1] https://archive.ics.uci.edu/ml/datasets/Iris

hyperplane easier in the transformed problem space. Some popular kernels are polynomial, radial basis function (RBF), hyperbolic tangent, etc.

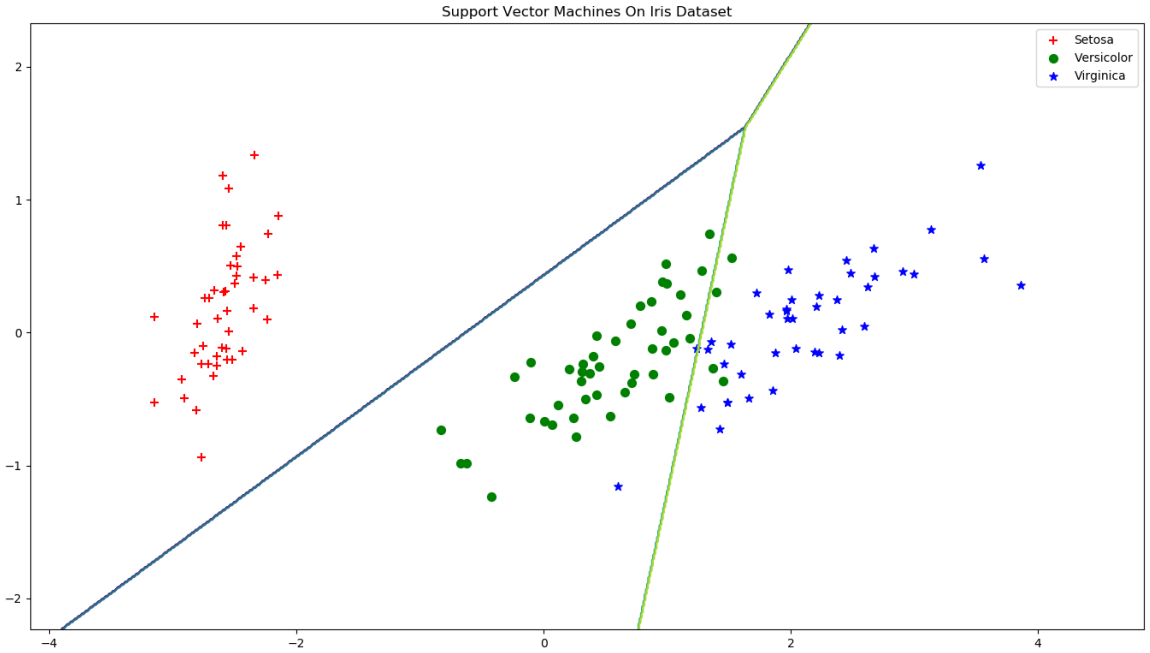


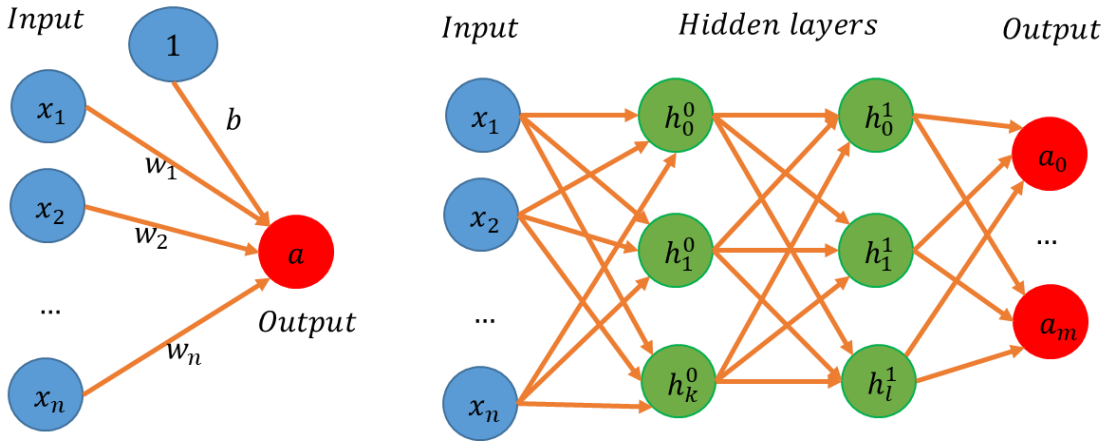
Figure 2.3 *The visualization of SVM classifier on the iris dataset*

2.3 Neural Networks

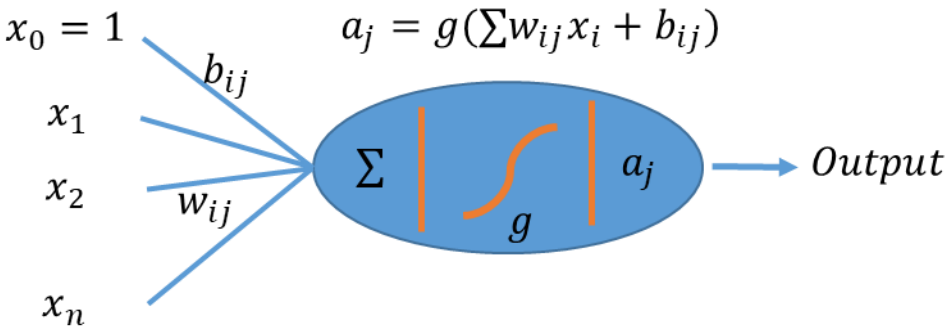
Artificial neural network (ANN) is the computation model that is stimulated by the neural systems in our brain (McCulloch & Pitts, 1943; Hebb, 1949). The first and simplest model of ANN is the perceptron (Rosenblatt, 1958). The overview model of the perceptron is presented on the left side of Scheme 2.1, while the multi-layer perceptron (MLP) is displayed on the right side. The perceptron, or a single neuron, takes multiple inputs but only has one output. The MLP model contains one or more hidden layers between the inputs and the outputs. Each hidden layer can also have multiple neurons. Also, there are one or more outputs in the MLP model. The perceptron is logistic regression alike. Both models calculate the linear combination of multiple inputs $\{x_i]$ with the weights w_{ij} and

bias b_j of the form $y_j = \sum_i w_{ij}x_i + b_j$, and then apply an activation function $g(y_j)$. The perceptron model uses a simple step function below to output the signal a_j:

$$a_j = 1 \; if \; y_j > 0, else \; a_j = 0$$



Scheme 2.1 *Examples of a perceptron (left) and a multi-layer perceptron (right) model*



Scheme 2.2 *A simple model of a neuron j, in which the set of input $\{x_i\}$ are linearly combined with the corresponding weights w_{ij} and bias b_{ij}, then apply through an activation function g, and fire an output a_j*

Scheme 2.2 displays the general computation outline of a neuron. Since the inputs are fed from the left side, such an MLP model is known as a feed-forward network. The outputs are computed at the first hidden layers, and then become the inputs for the next layer. That process continues until the final outputs are emitted from the output layer. Because of this feed-forward architecture, ANN can be used to learn any complex function

that maps the inputs to outputs. Therefore, analogous to different approaches that were introduced in section 2.2, ANN can be used for both classification/regression tasks. For the classification task, typically, the number of classes is used as the number of output neurons. A probabilistic activation function such as softmax[2] can be used at the output layers to transform the outputs into the interval $(0,1)$, which can be understood as the probability of such corresponding classes. Therefore, the prediction class is chosen from the output signal with the highest probability. For the regression task, at the output layer, the activation function is not usually used. In other terms, a 'linear' activation function is applied to the output, which just emits the output signal.

The most important part of the ANN is the optimization training process to adjust the multi-layer weight and bias parameters in the model to fit the inputs and outputs. Because the ANN model represents an approximation function from inputs to outputs, there are training errors, which is the difference between the outputs of the model with the actual outputs. There are two types of errors, called loss from now on. One type is the classification loss, which can be defined using the categorical cross-entropy function. Also, the other type is the regression loss, which can be defined using the mean squared error function. These losses can be considered as the functions of the weight and bias variables. Then, the training of the ANN model is converted into the optimization problem (minimize the loss on these variables). Such an optimization problem can be solved by using gradient descent with the help of backpropagation (Bryson Jr, Denham, & Dreyfus, 1963; Werbos, 1974; Rumelhart, Hinton, & Williams, 1986) algorithm. Backpropagation works by starting backward from the outputs layer, compute the gradient of the loss function respect

[2] The softmax function is originated from the Boltzmann distribution (1868), or Gibbs distribution (1902)

to the weights at the previous layer, update these weights correspondingly follow the gradients. Then it iteratively follows the chain rule to update the weights in the previous of the previous layer until reaching the first layer (input). Backpropagation and the variants of gradient descent algorithms are very important to neural networks, as well as deep learning because, without them, we could not train complex neural network models and witness the widespread adoption and development of deep learning nowadays.

2.4 Deep Learning Algorithms

This section will introduce the deep learning method as well as the two most important families of deep learning algorithms, which are Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNN).

2.4.1 Overview of deep learning

Compared with the traditional ANN model, also called "*shallow*" neural networks because of a limited number of hidden layers (one or two), deep learning is denoted by the introduction of the additional layers of depth than traditional ANN. In the 1990s and 2000s, even with the introduction of the backpropagation training algorithm from 1986, it is still challenging to train '*deep'* neural networks with multiple layers. That is because of the limitation of our computing capability at that time. However, with the increasing of computing power, especially the more powerful GPU, training deep neural networks is becoming more comfortable and faster. Also, the most crucial reason why deep learning becomes widely adopted is the more depth of the layer, the more representation levels of data will be learned (Marblestone, Wayne, & Kording, 2016). Each layer in the deep learning architecture works as a data abstraction function that transforms the input features into a higher level of representation in another space. Next, through the backpropagation

training, the best data representation model at each layer will be automated chosen to fit with the learning objectives. That is the reason why deep learning has a clear advantage over other machine learning methods because of the automated feature engineering and learning.

2.4.2 Convolutional Neural Network (CNN)

CNN is a subclass of deep neural networks that make use of convolutional layers. The idea of CNN is originated from the findings of Hubel and Wiesel in 1963 on the cat's visual cortexes containing receptive filed neurons that are arranged in layers to cover the entire visual field. The work of Hubel and Wiesel has inspired the creation of neocognition of Fukushima in 1980, and the time-delay neural network (TDNN) by Waibel et al. in 1987. A handwritten zip code number recognition system that used backpropagation with convolutions was introduced in 1989 (LeCun et al., 1989). It was until the 2010s that CNN made a breakthrough by its adoption on GPUs. The deep neural networks trained on GPU outperformed the previous methods on the MNIST dataset (Cireşan, Meier, Gambardella, & Schmidhuber, 2010). In 2012, the GPU-based CNN architecture design by Krizhevsky, Sutskever, and Hilton won the ImageNet challenge (Krizhevsky, Sutskever, & Hinton, 2012). From that time, CNN became widely used in image and video analysis, natural language processing, time series analysis, medical image analysis, etc.

The convolutional layer is the core element in a CNN architecture. It consists of a set of filters, also called kernels, that are applied like the sliding windows through the inputs to produce the outputs for the next layer. Figure 2.4 displays an example of a convolutional kernel moving across a 2D image. Also, Figure 2.5 presents a real example of a convoluted 2x2 filter on a sample 4x4 image region to produce a 3x3 output image. Besides the

convolutional layer, the pooling layer is also essential on CNN. A pooling layer also slides through the input like a window but performs an accumulation operation on a group of pixels belongs to the current windows. Such a popular pooling layer is the max pooling, which returns the max value amongst the elements in the window. Figure 2.6 displays an example of the max-pooling layer.

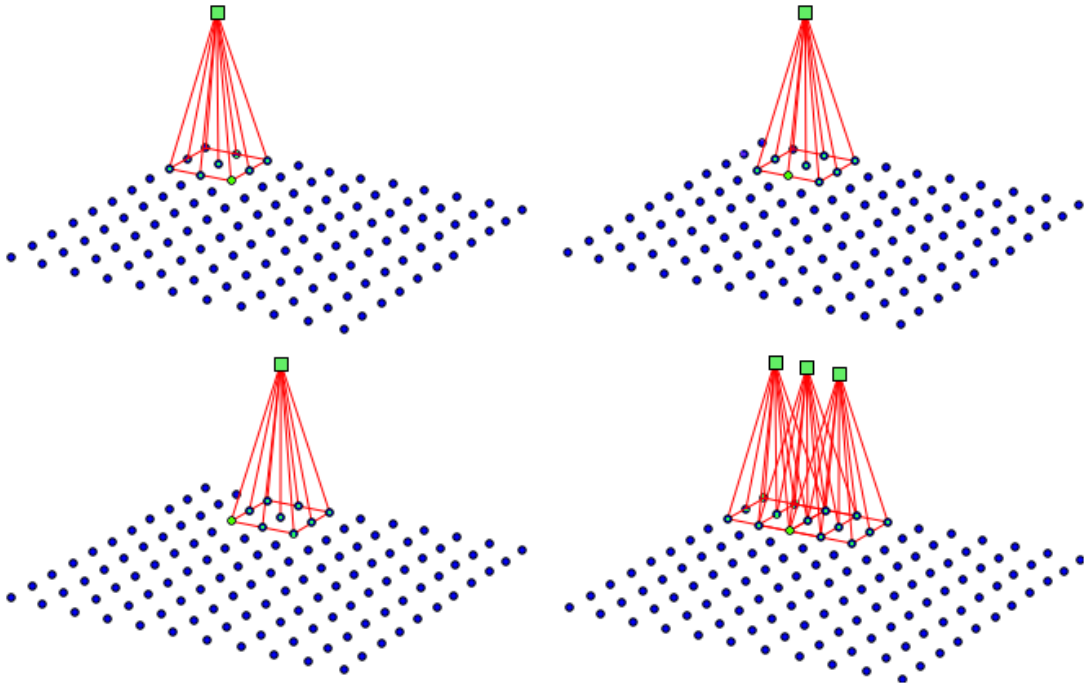


Figure 2.4 *An example of a filter moving across the image for convoluting*

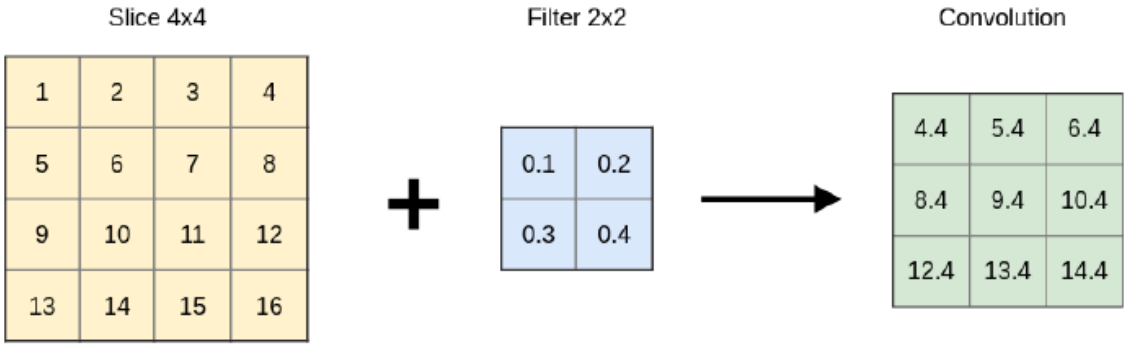


Figure 2.5 *A convolution example of a 2x2 kernel filter on a 4x4 image*

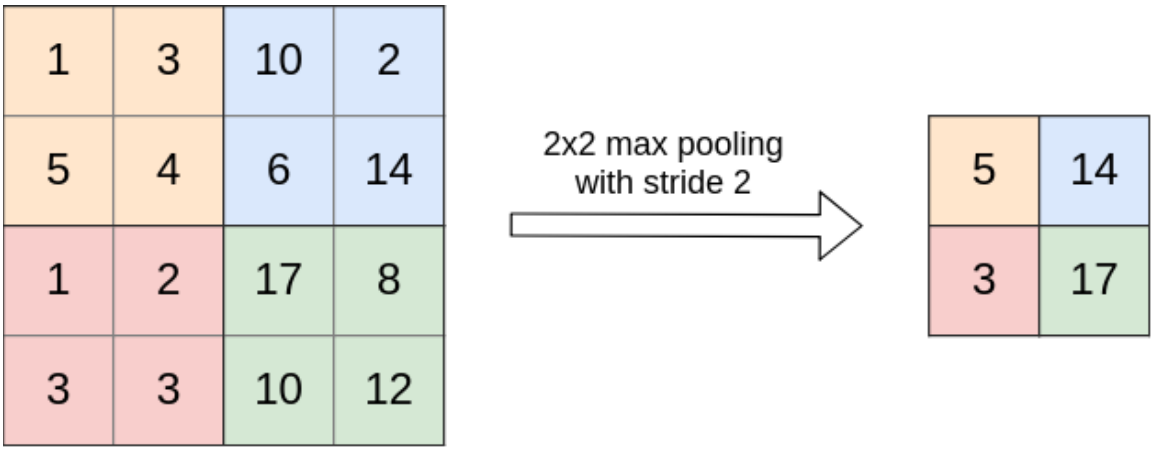


Figure 2.6 *An example of a 2x2 max pooling layer*

2.4.3 Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNN) (Hochreiter & Schmidhuber, 1997). RNN is a neural network class that is suitable for sequential data, such as time series, speech, text, etc. RNN is designed to maintain an internal state that can store information like a memory cell. Figure 2.7 describes the concept visualization of an RNN. At time step t, the network receives an input vector x_t, which can be the data inputs or the outputs of the previous layer. Next, it emits an intermediate output o_t while maintaining an internal state h_t. Then, both h_t, o_t will become the sequential input to feed to the next layer. The RNN update rules are described below:

$$a_t = b + W.h_{t-1} + U.x_t$$

$$h_t = \tanh(a_t)$$

$$o_t = c + V.h_t$$

$$y_t = softmax(o_t)$$

The three matrices U, V, W are the weight parameters of the RNN model and will be learned by backpropagation. RNN is limited in practice because of the vanishing and exploding gradients problem (LeCun, Bengio, & Hinton, 2015).

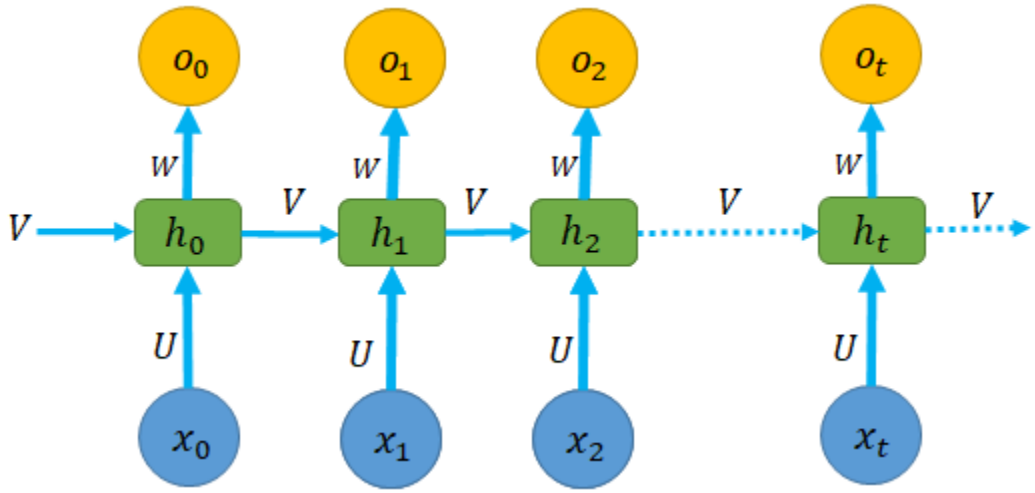


Figure 2.7 *The unfold structure of an RNN model*

Long Short-Term Memory (LSTM). Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) is a recurrent neural network method which prevents the problem of vanishing gradient. It also introduces long-term memory, which can capture the extended dependencies on previous information. LSTM makes use of regulated gates such as input gate, forget gates, and output gate to store or delete data in the cell state. LSTM can solve the problems of vanishing and exploding gradients of RNN. It can prevent the stored value from being repeatedly suppressed during training time by not using the activation function in the recurrent elements. The internal gates and data flow of a standard LSTM cell are shown in Figure 2.8. The update rules of LSTM model are described by the following equations:

$$i_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

$$f_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_f)$$

$$o_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_o)$$

$$\bar{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$c_t = f_t c_{t-1} + i_t \bar{c}_t$$

$$h_t = o_t \tanh(c_t)$$

The variables $i_t, f_t;\ o_t$ present the input, forget, and output gates, correspondingly. The gates values are reset either after batch feeding or after the entire sequence feeding.

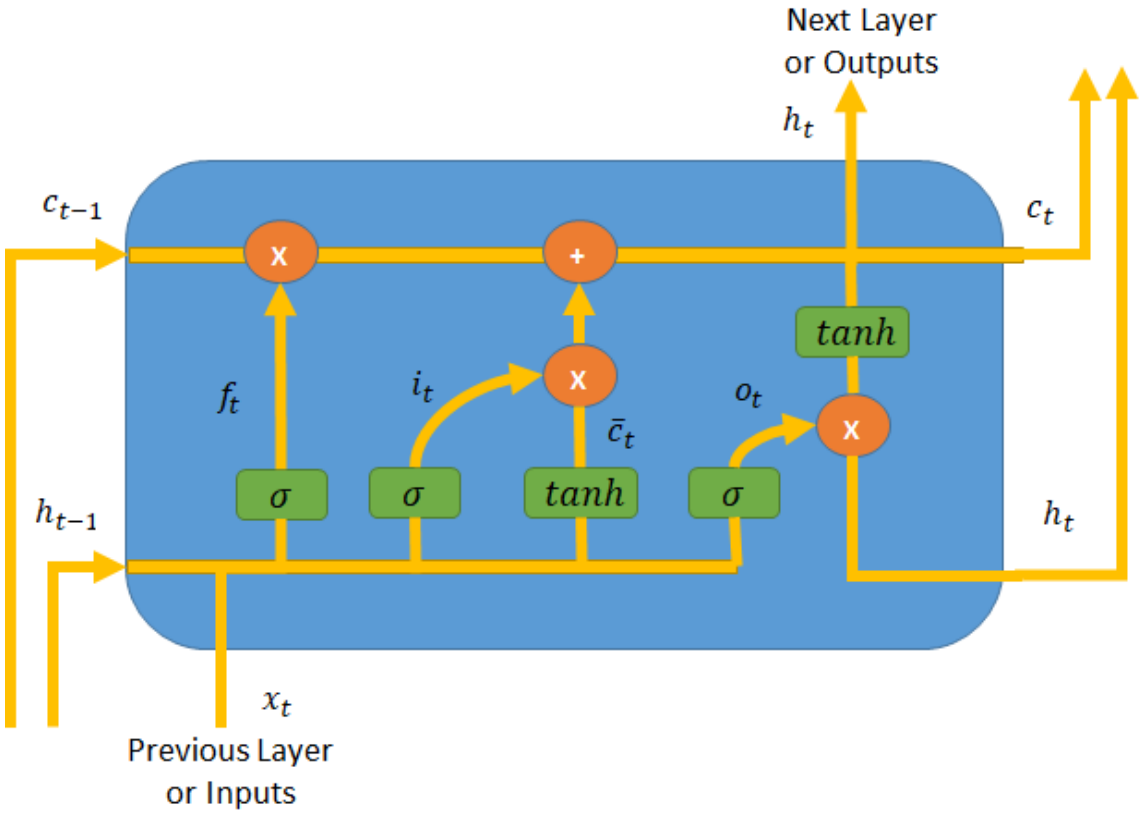


Figure 2.8 *The architecture of LSTM*

Note: redraw from (Hochreiter & Schmidhuber, 1997)

2.5 Reinforcement Learning

Reinforcement learning (Sutton & Barto, 2018; Kaelbling, Littman, & Moore, 1996) is a particular machine learning class that helps an artificial agent to master some

desired behaviors in a designed environment. Figure 2.9 displays the simplified and full principle of reinforcement learning. In the simplified version, at a timestep t, the agent has its own policy (controller) that executes an action a_t in the environment, and in turn the environment delivers feedback which contains the new observation state s_t and a reward signal r_t. The agent's policy will generate further appropriate action a_{t+1} for the next timestep based on the new inputs. The policy is designed to maximize the accumulative reward received from the system over time.

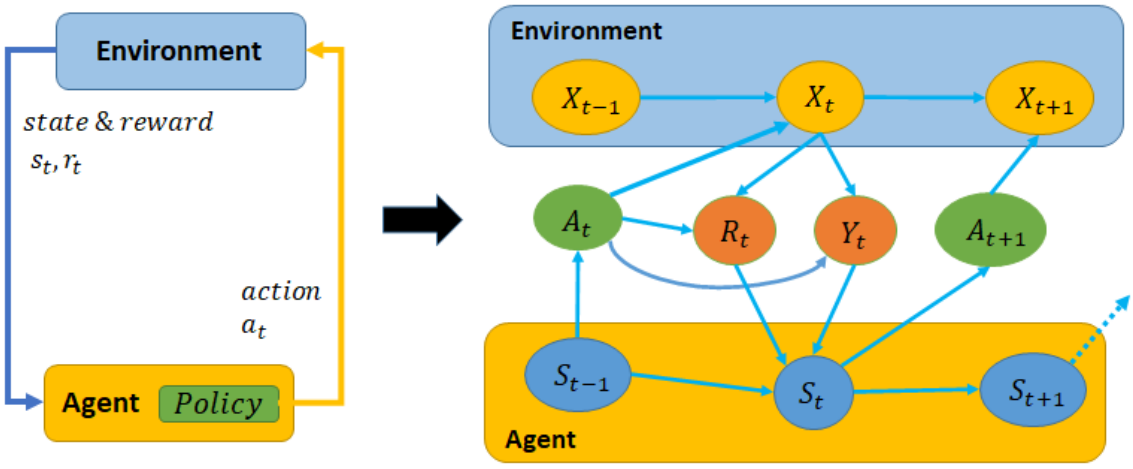


Figure 2.9 *The overview principle of reinforcement learning*

In Figure 2.9, X_t is the environment's internal state; S_t is the internal state of the agent; R_t and Y_t are the reward signal and the observation state the agent received from the environment, and A_t is the action that the agent sends to the environment. Next, we can define the expected of accumulated discounted reward: $E[R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots] = E[\sum_{t=0}^{\infty} \gamma^t R_t]$, in which $0 \leq \gamma \leq 1$ is the discount factor. The discount value is used to value recent immediate rewards more than future rewards to force the system to converge faster. If the agent receives the full observation state, which means $Y(t) = X(t)$, we have

a Markov Decision Process (MDP) (Bellman, 1957) model. However, in reality, the agent can only see part of the full observation state of the environment, and then we have a Partially Observable MDP (POMDP) (Kaelbling, Littman, & Cassandra, 1998).

MDP. This framework defines the model for a fully observable reinforcement learning problem (Bellman, 1957; Van Otterlo & Wiering, 2012). The model identifies four components: $< S, A, P, R >$. $S = \{s_t\}$ denotes the set of possible environment states. $A = \{a_t\}$ denotes the set of possible actions. P is the function of state action transition that defines the conditional probability from a current state s, and an action a.to transform to a new state s'. $R = \{r_t\}$ denotes the reward function. Now we can define the state-value function $v_\pi(s)$ which defines the expected sum of rewards if we follow the policy π starting from the state s_t:

$$v_\pi(s) = E_\pi[\sum_{j=0}^{T} \gamma^j r_{t+j+1} \mid s = s_t]$$

The action-value function $q_\pi(s, a)$ is the expected sum of rewards if we follow the policy π, starting from the state s_t, and taking the action a:

$$q_\pi(s, a) = E_\pi[\sum_{j=0}^{T} \gamma^j r_{t+j+1} \mid s = s_t, a = a_t]$$

We have the relation that $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$, in which $\pi(a|s)$ is the probability that the policy π selects the next action a given the current state s.

Bellman equation (Bellman, 1957): This is the foundation equation that helps researchers develop the algorithms for reinforcement learning. Bellman equation defines that the state-value function $v_\pi(s)$ is the intermediate reward of that state plus the discounted state-value of the next state in case the optimal action is performed:

$$v_\pi(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) v_\pi(s')$$

Based on the above equation, Bellman proposed two algorithms: value iteration and policy iteration that can be solved by dynamic programming to find the optimal policy. Another fundamental approach to address reinforcement learning is the Monte Carlo method. The idea of this approach is the state-value function of a state s is approximated by the mean value of the sampling state-value of such state over many episodes: $\bar{v}_\pi(s) = \frac{1}{N}\sum_{i=1}^{N} v_{i,s}$. Therefore, in this approach, we repeatedly perform actions through a lot of episodes to collect sample sequences of states and rewards. These samples will be used to estimate the state-value function, and finally, the optimal policy can be determined.

Temporal-difference (TD) learning (Sutton, 1984; Sutton & Barto, 2018; Bradtke & Barto, 1996). This algorithm replaces the Bellman equation by an approximation update rule that can help the agent to learn a better policy. The update rule is $v_\pi(s) \leftarrow v_\pi(s) + \alpha(r + \gamma v_\pi(s') - v_\pi(s))$, with $0 < \alpha \leq 1$ is the learning rate, while follow the policy π that change the current state s to s' with a reward r.

Sarsa. This algorithm introduced an update rule to learning the action-value function $q_\pi(s,a)$, instead of the state-value function $v_\pi(s)$, using the action-value function value of the next state s' and next action a' (often chosen from ϵ-greedy policy):

$$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha[r + \gamma q_\pi(s',a') - q_\pi(s,a)]$$

Q-learning (Watkins & Dayan, 1992). Similar to Sarsa algorithm, the update rule is: $q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha[r + \gamma \max_a q(s',a) - q_\pi(s,a)]$. The main difference with Sarsa is, instead choose the next action from ϵ-greedy policy, the optimal action that $\max_a q(s',a)$ is selected. With a finite set of states and actions, Q-learning can store the

estimated action-value in a table for lookup. However, for continuous state-space or action-space, the function approximation technique can be used. In this case, ANN can be used as an action-value function approximator to generate the probability to select the optimal action, given a state.

Policy search (Sutton & Barto, 2018): Similar to the function approximation method, this kind of algorithm is designed for large or continuous state and action space. The idea is represented the policy π by a parametric form on a set of parameters θ, such as $\pi_\theta(s)$. Then the policy search algorithm optimizes the parameters θ to improve the policy. An example is using the parameterized Q-functions to represent the policy π: $\pi_\theta(s) = \max_a \hat{Q}_\theta(s, a)$. Given a current state s, this algorithm selects the next action with the highest predicted value of $\hat{Q}$. In the family of policy search algorithms, a stochastic policy $\pi_\theta(s, a)$, that indicates the probability of selecting an action a given the state s, is often used. The softmax function is a popular representation method in this case. Policy gradient methods are one of the popular types of the policy search algorithm. This group of algorithms uses neural networks to approximate the policy, and use gradient descent to optimize the policy π_θ (update the weights and biases in the neural networks model) by optimizing the policy performance of a single episode function $J(\theta) = v_{\pi_\theta}(s_0)$. Here, s_0 is the initial state of an episode and $v_{\pi_\theta}(s_0)$ is the expected sum of reward from s_0 to the end of the episode.

2.6 Deep Reinforcement Learning

The term deep reinforcement learning is coming from the combination of deep learning and reinforcement learning (François-Lavet, Henderson, Islam, Bellemare, & Pineau, 2018). This type of technique can solve a lot of complicated decision-making

problems, especially the ones with high dimensional or continuous state-action spaces, in games, robotics, healthcare, finance, etc. There are two most recent breakthrough results in games that mark the revolution of deep reinforcement learning. The first one is the design of a computer agent that beats the human-level in Atari games, just by observing directly the image captured from the game (Mnih et al., 2015). The second one is the creation of mastering the game of Go named AlphaGo Zero and Alpha Go system, which can defeat the human world champion (Silver et al., 2016).

Deep Q-learning or Deep Q-network (DQN) (Mnih et al., 2015)**:** This is the variant of the Q-learning algorithm. Based on the neural fitted Q-learning (NFQ) algorithm proposed by Riedmiller in 2005, Mnih et al. introduced two heuristics to deal with the instabilities of the NFQ algorithm: the target Q-network and the experience replay memory. In the NFQ algorithm, the Q-values are parameterized using ANN in the form of $Q_{\theta_k}(s,a)$, in which θ_k is a set of parameters at the k^{th} iteration. The target value function at this iteration is estimated by the following equation: $V_k^Q = r + \gamma \max_{a' \in A} Q_{\theta_k}(s',a') \quad (Eq.2.6a)$. Variants of stochastic gradient descent are typically applied to update θ_k by minimizing the following loss: $L_k = \left(Q_{\theta_k}(s,a) - V_k^Q\right)^2$. The following equation defines the parameters updating rule: $\theta_{k+1} \leftarrow \theta_k + \alpha \left(V_k^Q - Q_{\theta_k}(s,a)\right) \nabla_{\theta_k} Q_{\theta_k}(s,a) \quad (Eq.\ 2.6b)$, in which α is the learning rate in the range of $(0,1)$.

Mnih et al. replaced the target Q-network V_k^Q by $Q_{\overline{\theta_k}}(s',a')$, where $\overline{\theta_k}$ is similar to the parameters θ_k, but will be updated in a slower frequency using the assignment $\bar{\theta}_k = \theta_k$. This idea is keeping the target values fixed while evaluating the Q-values to prevent instabilities. Another important heuristic in DQN is the experience replay memory. The

experiences are collected by the ϵ-greedy policy and put into the replay buffer with the maximum capacity of N. During each training iteration, mini-batch samples are selected from the replay buffer to update the parameters of the Q-networks. This technique is proved to reduce the variance, more sample efficient compared to other on-policy learning methods. There are some useful variants of DQN, such as Double DQN (Van Hasselt, Guez, & Silver, 2016), Dueling DQN (Wang et al., 2015), DQN with Prioritized Experience Replay (PER) (Schaul, Quan, Antonoglou, & Silver, 2015).

Policy Gradient (Sutton & Barto, 2018)**:** This is a type of policy-based algorithm. In this kind of method, the policy is parameterized by the set of parameters θ in the form of $a_t \leftarrow \pi_\theta(s_t)$. Based on directly the observations s_t, the policy determines the corresponding optimal action a_t from a probability distribution. Next, if following such the policy, we have a trajectory sample τ containing a sequence of state-action $< s_1, a_1, s_2, a_2, \dots, s_T, a_T >$ in an episode of length T. Then the probability of such a sequence is:

$$p_\theta(\tau) = p_\theta(s_1, a_1, s_2, a_2, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

Now, the optimal policy with the parameters θ is the one that satisfies this equation: $\max_\theta E_{\tau \sim p_\theta(\tau)}[\sum_t r(s_t, a_t]$. Such the parameters can be optimized by following the gradient as

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\, Q_\theta(s_{i,t}, a_{i,t})$$

Then, the delta step magnitude to update the parameters can be calculated by $\Delta(\theta) = \alpha \nabla_\theta J(\theta)$, which travel to the direction that the accumulated rewards increase. The

basic principle of policy gradient is we increase the likelihood $\log \pi_\theta(s, a)$ of the actions that lead to better rewards. Some approaches try to limit the policy change in a trust region to improve the convergence of the model. Such methods are Trust Region Policy Optimization (TRPO) (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015), Proximal Policy Optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017),.

CHAPTER III – INFORMATION DIFFUSION ON SOCIAL NETWORKS

This section provides essential background information and related works to help better understanding the goals of this thesis. This section provides an overview of previous works in information diffusion on social networks.

3.1 Introduction to Information Diffusion

Communication is a vital part of human beings. The widespread information exchange phenomenon will not be possible nowadays without the recent development of our communication channels, primarily Internet-based services such as online social networks, weblogs, etc. Since the introduction of online social networks (OSNs), their usage has been developed in many aspects, and make them one of the most critical communication channels of human beings for political / promotion campaign, opinion formation, viral marketing, etc. Nowadays, online social networks can connect hundreds of millions of users worldwide to exchange information and therefore create a massive volume of information sharing sources. Understanding the principles of information flow on social networks can give us a lot of benefits. Such benefits are improving business performance (viral marketing, promotion campaign, etc.), cybersecurity (preventing planned terrorist attacks, information formation, etc.), opinion mining, etc. Motivated by the above advantages, social network analysis has become a popular research topic.

Social networks provide users the platform to connect and contribute to the information diffusion (or information diffusion or dissemination) by sharing and spreading information. Bakshy et al. have summarized the important role of social networks in increasing the diffusion process of information and viewpoints (Bakshy, Rosenn, Marlow, & Adamic, 2012). Also, information diffusion is a vital phenomenon of online social

networks. Therefore, understanding the patterns and behaviors of information propaganda on social media can help predict the status, reputation, and public sentiments of different events. Different models or techniques have been formed to describe the information diffusion in social networks, analyze and maximize the influence factors, determine how users interact on the systems, community detection, topic modeling, sentiment analysis, and volume prediction, etc. As summarized by Guille et al., all of these research on information diffusion are trying to answer the following questions: a) what information or topics are popular and propagate more than the other, b) how and why information can diffuse in different ways, c) how the information will distribute in the future, d) what are the import factors of social network that control the information spreading process (Guille, Hacid, Favre, & Zighed, 2013).

The following sections will discuss in detail about different research topics in information diffusion on social networks.

3.2 Information Diffusion Modeling

According to Haralabopoulos, sociologists have studied social ties and information flow in traditional social networks even before the rise of online social networks (Haralabopoulos, Anagnostopoulos, & Zeadally, 2015). The idea of information flow, which captures the spreading sequence of mass media, opinion leaders, and consumers, was introduced from the 1940s when the authors analyzed the presidential campaign (Lazarsfeld, Berelson, & Gaudet, 1944). Later, Kaltz and Lazarsfeld proposed the theory of the "two step flow of communication" in the 1970s (Katz & Lazarsfeld, 1970). The introduction of email in the mid-1970s and the social networks in the 1990s, have made an enormous transformation in the way we communicate and exchange information (Bakshy,

Rosenn, Marlow, & Adamic, 2012) or do business such as viral marketing (Jurvetson & Draper, 1997). Modeling information diffusion is an important research topic that attracts a lot of attention. Guille et al. have described two fundamental aspects to model the information diffusion process: a) the activation sequence: the diffusion graph and b) the spreading cascade: the evolution of the diffusion rate. In 2013, Guille et al. have summarized and classified different information diffusion modeling methods as explanatory models and predictive models (Guille, Hacid, Favre, & Zighed, 2013). In 2017, Li et al. revisited the problem and extended their works with the splitting explanatory models into epidemics models and influence models, and add other relevant information diffusion modeling methods (Li, Wang, Gao, & Zhang, 2017).

Information diffusion on social networks displays intelligent and complex behaviors that depend on many significant factors such as the topics (hashtags on Twitter), the underlying network structure or communities, the sentiment of the discussed topics, user location, and the effects of influencing, etc. Different approaches to model information diffusion often consider these essential factors. In general, there are two types of information diffusion modeling methods that are discussed in the subsequent subsections of 3.2.1 and 3.2.2.

3.2.1 Structure-based Diffusion Modeling

The early type of information diffusion models was based on the graph (Granovetter, 1978; Guille, Hacid, Favre, & Zighed, 2013). In these models, while the nodes stand for the users, the edges correspond to the relationship between them. The explicit link is the friend/follower connection between two nodes. Also, different information diffusion models try to study other implicit relationships, such as influencing,

retweeting, etc. Network structure has been proved that performs a crucial role in information flow, social capital, and knowledge transfer, in which network range and social links are essential than the connection strength among two user nodes (Reagans and McEvily 2003). Also, another research showed that the topic popularity is correlated with the number of followers/friends who have interacted with that topic (Bacstrom et al. 2006). A quantitative study on the follower-following topology analysis that discovers different characteristics of information sharing on Twitter, including small effective diameter and low reciprocity (Kwak, Lee, Park, & Moon, 2010).

Some approaches are using a similar model to the process of spreading diseases or epidemics to model information diffusion. The process that information diffuses between nodes is considered analogous to virus transmission from infected to susceptible users. Some fundamental epidemic models are Susceptible Infected (SI), Susceptible Infected Susceptible (SIS), Susceptible Infected Removed (SIR), and Susceptible Infected Removed Susceptible (SIRS) (cited in (Li, Wang, Gao, & Zhang, 2017)). Based on the above models, researchers develop new epidemic-like information diffusion models in social networks. Such models are Susceptible Exposed Infected Removed (SEIR), Single layer-SEIR (S-SEIR), Susceptible Contacted Infected Removed (SCIR), infected recovery SIR (irSIR), fractional SIR (FSIR), and Emotional Susceptible Infected Susceptible (ESIS) (cited in (Li, Wang, Gao, & Zhang, 2017)). These models try to study the distribution of the susceptible (S), infected (I), exposed (E), contacted (C), and removed (R) nodes. However, these models can fit into a particular dataset from the networks but are found difficult to adapt to the dynamics change of such real networks.

Some other approaches model the information diffusion based on the influence factor. The information spreading cascade is determined by the underlying network structure and the dynamics of the diffusion rate. The global diffusion rate will be decided by the number of existing and newly affected nodes at a time. Independent Cascade (IC) and Linear Threshold (LT) (Granovetter, 1978; Shakarian, Bhatnagar, Aleali, Shaabani, & Guo, 2015) are two of the early models in this category. Figure 3.1 displays examples of IC and LT models. In the IC model, each edge in the network has an associated diffusion probability. In each iteration, the source node tries to activate the target node with such a defined probability. On the other hand, the LT associates an influence degree with each edge and an influence threshold with each node. Then, the inactive user node can be activated (affected) at each iteration, if the total of influence degrees from its neighbors is more significant than its influence threshold. Based on these two basic models, many variants of them have been introduced such as asynchronous independent cascades (AsIC), asynchronous linear threshold (AsLT) proposed by Saito et al., 2011, Time-Based Asynchronous Independent Cascades (T-BaSIC) model (cited in Guille at al., 2013).

In 2010, Gomez et al. introduced a model named NETINF (cited in Guille et al., 2013) that defines a probability that a node transmits information to neighbor nodes. Next, they introduced an extended model named NETRATE (cited in Guille et al., 2013) in 2011 that uses a probability density function of infection time and transmission rate to determine the likelihood that a node will infect neighbor nodes. Later, in 2013, Gomez et al. introduced the INFOPATH model, which uses stochastic gradient descent to estimate the structure and temporal features (cited in Guille et al., 2013).

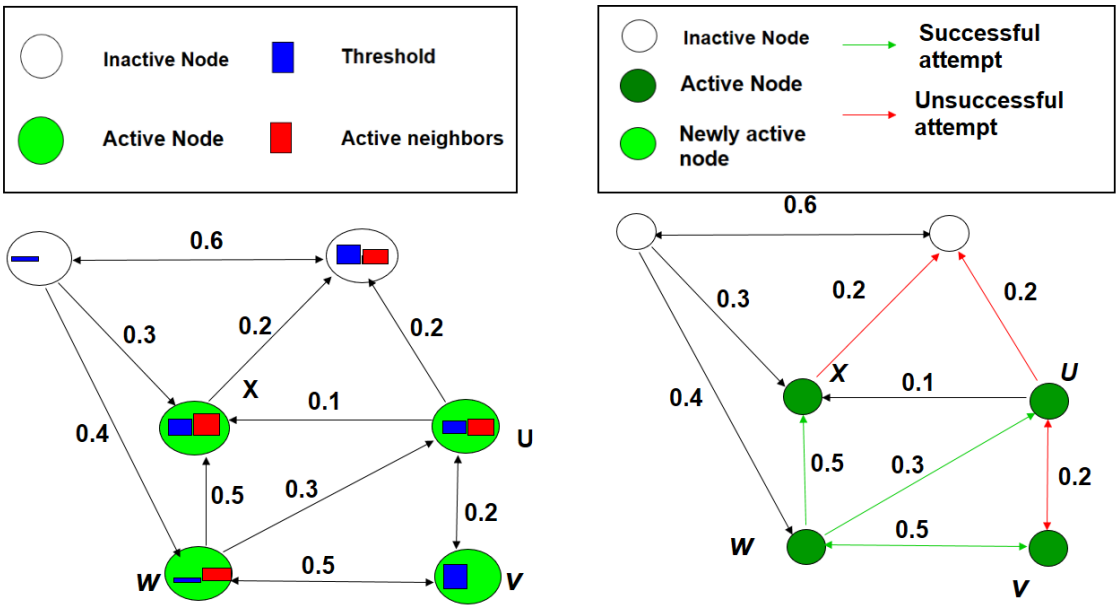


Figure 3.1 *The examples of Linear Threshold model (left) and Independent Cascade model (right)*

3.2.2 Content-based Diffusion Modeling

The discussion topics and the content in the messages are essential factors to determine the temporal dynamics of the information diffusion process. Different topic modeling methods on social networks have been introduced such as Labeled-LDA (Ramage, Hall, Nallapati, & Manning, 2009), Author-Topic (Hong & Davison, 2010) model, Twitter-LDA (Zhao et al., 2011) model, the Word Network Topic Model (WNTM) (Zuo, Zhao, & Xu, 2016), etc. There are also research works that model the relation between information propaganda and discussion topics. An example is an information diffusion modeling framework in social networks using linear multivariate Hawkes processes and the LDA topic model (Pinto and Chahed 2014).

Besides the structured-based influence modeling methods in the previous section, some approaches avoid such network structure-dependent by targeting the individual nodes. One of the famous techniques is the LIM model developed by Yang and Leskovec

in 2010. They model the global diffusion volume using a linear combination of individual users' influence function with the user-topic influence indicator matrix. The influence function of a user is modeled as a fixed-length time series vector. They used the Reflective Newton method to solve the non-least square problem to find the influence vector for each user. This model can predict the global diffusion volume in future timesteps. Later, this model was extended to take into account the external influence factor (Myers, Zhu, & Leskovec, 2010). There are also other approaches in this direction such as the simplified SIS model of Leskovec et al., the Partial Differential Equation (PDE) method of Wang et al. to predict the diffusion rate of a node on a topic (cited in Guille et al., 2013), the topic-level direct influence mining method (Liu, Tang, Han, Jiang, & Yang, 2010).

The sentiment of the discussion topics can alter the diffusion rate of such topics over time because the users often express their own opinions or emotions on a specific topic. Therefore, sentiment analysis on social networks, especially Twitter, has become an exciting research topic (Tang et al., 2014; Wang, Varghese, & Donnelly, 2016). For example, a framework, containing dictionary-based and machine learning-based approaches, that analyze and visualize the geographical public sentiment, was proposed (Nguyen, Varghese, & Barker, 2013). Also, in 2015, a study that quantified the effect of sentiment during the information propagation process, found out that negative posts diffuse faster than the positive ones. Still, positive ones get more shares and favorites (Ferrara & Yang, 2015). They also showed different temporal sentiment-diffusion patterns.

3.3 Information Diffusion Prediction Models

Predictive models are designed to forecast how information propaganda processes would spread the information over a given network at future timesteps. Typically, these

models should learn to anticipate the past diffusion traces from temporal and spatial points of view. Information diffusion prediction models are essential because, in some cases, “bad” or misleading information propaganda should be stopped in the network.

All the information diffusion models reviewed in the previous section (3.2) can be used as the foundation of predictive models. The fundamental flowchart will be, first collecting historical diffusion dataset from the real networks and then derive the optimal parameters for these diffusion models. Next, the predictive models will be continuously adaptive to the temporal dynamic changes of the networks. Some early approaches to build the diffusion prediction models are based on the fundamental IC and LT models. Also, many influence maximization approaches are developed from the above models, which try to discover the seed nodes that can maximize the influence effect to gain a broader scope of information propaganda. Such approaches are the IC-based model to predict diffusion probability using the Expectation-Maximization algorithm (Saito, Nakano, & Kimura, 2008), the influence maximization algorithm of Wang et al. and Jung et al., the ASIM algorithm of Arora et al., etc. (cited in Li et al., 2017). Similarly, different LT-based models have been adapted for prediction purposes such as the DRUC model of Lagnier et al., the heuristic activation threshold algorithm of Chen and Yitong (cited in Li et al., 2017). Also, as mentioned before, the use of the Linear Threshold Model of Yang and Leskovec (2010) for diffusion prediction tasks.

3.4 Influence Analysis of Social Networks

Recently, influence analysis and influence maximization (Kempe, Kleinberg and Tardos 2003) have become two important research topics in social network analysis. It is observed that during the information diffusion process of a topic, the influence rate of each

user node is different from each other. As mentioned in section 3.2.1, early approaches of influence analysis are based on the graph to model the influence/activation parameters between user nodes such as Linear Threshold (LT), Independent Cascade (IC), AsLT, AsIC, NETINF, NETRATE, INFOPATH, etc. However, these models could not adapt well to the temporal dynamics of the underlying networks in real life.

Topic-related influence analysis is also another direction of this research domain. An example is the generative graphical model to determine direct and indirect influence at the topic level (Liu, Tang, Han, Jiang, & Yang, 2010). The direct influence was determined by the link and text contents of each node. Next, the indirect influence was derived from a topic-related influence diffusion model. Later, they defined an aggregation algorithm of both types of influence to study the relationship between user behaviors and topic-level influence. In 2012, another framework was introduced to evaluate the social impact and the similarities of related users in a heterogeneous network (Wang, Hu, & Yu, 2012). Besides, in 2014, a topic-specific influence analysis study was conducted on the microblogs (Bi, Tian, Sismanis, Balmin, & Cho, 2014). This study introduced a useful Followship-LDA model that defined the content-based following relationship and content-independent following relationship. The content-based following relationship is established if two users follow each other and regularly tweet/retweet on the same topic. On the other hand, the content-independent following relationship is based on the popularity of a user node, so they have a lot of friends/followers.

Yang and Leskovec performed one of the remarkable researches of influence analysis in 2010. They introduced a Linear Influence Model (LIM) that models the global diffusion volume as a linear combination of individual influence function of a node in a

social network. They used a fixed-length vector to model such user influence and used the Reflective Newton method to find these user vectors. Myers et al. extended the LIM model with the introduction of external influence factors. They pointed out that in their studies, about 71% of the information was diffused over Twitter (by internal factors). In comparison, external factors contributed to the remaining 29% of diffusion (Myers, Zhu, & Leskovec, 2010). In 2013, an approach to verify the existence of social influence in the various topology of social networks was proposed, and then a computational model to quantify social influence was introduced (Sun & Tang, 2013).

3.5 User Network Interaction and Topic Diffusion on Social Networks

Hashtags are commonly used on Twitter because they are an explicit way to set up the discussion topics. Hashtags are tagged by a group of users in their tweets to help them efficiently to keep track of multiple concurrent discussion topics. Moreover, hashtags are also the mechanism to help Twitter users find and join dynamic communities. Communities on Twitter are implicitly established by user groups who have related topics of interest. However, using hashtags for topic detection and modeling is not enough. A previous study has revealed that only 5% of tweets include a hashtag (Boyd, Golder, & Lotan, 2010). Therefore, efficient topic modeling models need to be developed because not only such a good model can help understand the pattern of information diffusion but also help us in user/friend recommendation or link prediction.

Topic modeling. In natural language processing, a topic model is used to represent the set of possible topics that exist in a document collection. Therefore, in text-mining, topic modeling is widely used to discover the hidden semantic aspects of the documents. One of the most popular topic modeling methods is Latent Dirichlet Allocation (LDA)

(Blei, Ng, & Jordan, 2003), which extracts the sparse Dirichlet prior distributions from the distributions of document-topic and topic-word. Because the tweets on Twitter are short and imbalanced, the LDA model and its extensions fail to capture the appropriate topic models. Therefore, multiple extensions of the LDA for the Twitter platform have been developed to handle the above problem. The Twitterrank (Weng, Lim, Jiang, & He, 2010) algorithm or Author-Topic (Hong & Davison, 2010) model, aggregated all user's tweet to make a long enough document for traditional LDA. Also, the Twitter-LDA (Zhao et al., 2011) model, introduced a user-topic distribution θ^u of a user u, a topic distribution θ^t of a topic t, and a Bernoulli distribution π for background. This model assume that a tweet is associated with a single topic. The authorship of a tweet is the process of selecting the topic t from θ^u, followed by a Bag of Words (BoW) selection either from θ^t or π. Besides, the Word Network Topic Model (WNTM) (Zuo, Zhao, & Xu, 2016), collected the word co-occurrence matrix to form the topic modeling distributions.

User recommendation. As suggested by Weng et al., the following relationship and retweeting behaviors of users are closely correlated to the user-topic distributions and how users interact on the social networks (Weng, Lim, Jiang, & He, 2010). Therefore, there are graph-based approaches (Armentano, Godoy, & Am, 2011) that consider the friends and followers network graphs of the users to recommend the potential followers to a similar user (Kywe, Lim, & Zhu, 2012). Moreover, there are other feature-based methods such as the structural user recommendation (Golder, Yardi, Marwick, & Boyd, 2009) method, the weighted content-based user recommendation (Garcia-Gavilanes & Amatriain, 2010) method, the Twittomender (Hannon, McCarthy, & Smyth, 2011) user recommender system, and the sentiment-based user recommendation (Gurini, Gasparetti, Micarelli, &

Sansonetti, 2013) method. Also, topic-models based user recommendation methods have been introduced such as the weighted combination topic models (Ramage, Dumais, & Liebling, 2010) of TF-IDF and Labeled-LDA (Ramage, Hall, Nallapati, & Manning, 2009), the adapted user-level LDA topic models (Pennacchiotti & Gurumurthy, 2011).

Retweet prediction. Retweet/reply behaviors are essential factors that affect the information diffusion rate. Therefore, there is research to analyze and predict these behaviors. One of the first approaches is building the user profiles for similarity matching. If a source profile and a target profile have such similar patterns, they are supposed to have the same tweet/retweet behaviors. There are some examples of this kind of approach, such as the Wikipedia-linked tweets user profiling method (Macskassy & Michelson, 2011), the TF-IDF bag of words user profiling method (Xu & Yang, 2012). Some other approaches use the features from the users' contents to build the retweet prediction model (Firdaus, Ding, & Sadeghian, 2016). Such an approach is the study of information strategies of users based on the set of features, including Linguistics, Freshness, Trustability of Information, and Interest Matching (Nguyen, Tan, Ramanathan, & Yan, 2016). The study also introduced a useful similarity model of the TF-IDF weighted Bag of Words to predict retweeting behavior.

CHAPTER IV – COLLECTIVE TASK LEARNING ON SWARM ROBOTICS

This section reviews some related works in swarm robotics and solving collective task learning problems in swarm robotics.

4.1 Introduction to Swarm Robotics Research

Our nature has given us a lot of wonderful examples of a powerful and efficient society of insects such as ants, honey bees, termites, etc. (Şahin, 2004). In these societies, many simple individuals can interact and coordinate their actions in a collective manner that can accomplish difficult tasks, which cannot usually be handled by an individual. Such a phenomenon has inspired the creation of the research domain of swarm robotics, or collective-robots, or distributed robots, etc. Swarm robotics can be considered as a combination of a multi-robots system with swarm-intelligent capabilities, which simulate the social structures and interactions in the system (Tan & Zheng, 2013). Three characteristics of a multi-robot system to be considered swarm robotics are robustness, flexibility, and scalability (Şahin, 2004; Brambilla, Ferrante, Birattari, & Dorigo, 2013). Robust is the ability to continue operation even there are failures in part of systems or individuals. Robustness achieves by the factors of redundancy, decentralized sensing & coordination, and simplicity of swarm robotics. Also, the flexibility of the swarm robotics system comes from the utilization of individual roles and coordination mechanisms to deal with different kinds of tasks and adapt to the change in the environments. Finally, the scalability property of swarm robotics denotes the stability in the swarm performance, disregard the change in the size.

Design. Some agreement criteria distinguish the swarm robotics research with other disciplines (Şahin, 2004; Brambilla et al., 2013). The first criterion requires the robots

in swarm should be autonomous and decentralized. These robots have physical bodies, interact with surrounding worlds through sensors, actuators, etc., and can operate on their own. The second condition is the swarm robotics should contain a large enough number of individual robots for scalability. Next, for the flexibility property, the swarm should consist of a few homogeneous groups of robots. The major of individual robots should have similar design, structure, and capabilities for mass production and flexibility. The performance of swarm robotics comes from the collective behavior of a whole group, rather than the outstanding performance of an individual. Lastly, robots in swarm often have limited local sensing, communication, and computing/memory capabilities.

Applications. Several types of research have pointed out the promising applications of the swarm robotics system (Şahin, 2004; Dorigo et al., 2014; Bayındır, 2016). Because of the distributed nature of the swarm robotics, it is suitable for applications that cover a region such as surveillance, environmental monitoring, etc. Also, it is used to execute dangerous tasks for humans, such as clearing the mining field, collecting toxicity chemicals, deep-water exploration, etc. The scalability and flexibility properties of the swarm robotics make it suitable for applications that require the flexibility in the scale such as sell-assembly, or redundancy such as communication beacons, etc. In 2016, a broad study presented by Bayındır, summarized the design models, related works, models, and performance for different kinds of tasks in swarm robotics such as aggregation, flocking, foraging, navigation, deployment, and task allocation, etc.

There are different directions of research in the domain of swarm robotics, such as design, analysis, and collective behaviors (Dorigo, Birattari, & Brambilla, 2014). The collective behavior research can be classified into multiple groups. Spatially organizing

behavior research is conducted to design a good strategy to coordinate the robots to achieve predefined spatial requirements or configurations. This kind of research is used for applications like aggregation or coverage. Navigation is another vital research to help the agents reaching the target with limited communication in applications such as area exploration or goal searching. Collective decision making is another necessary research that contributes to coordinate the agents and interact with the environment to accomplish the goal. Such critical applications of collective decision making are consensus, task allocation, motion planning, collective transportation. Collective decision-making problems are widespread tasks in swarm robotics. Collective decision making (Okubo, 1986; Kao, Miller, Torney, Hartnett, & Couzin, 2014), is defined as members in a group cooperatively agree on some options without centralized control. Collective decision making (Valetini, Ferrante, & Dorigo, 2017) has typically been applied in consensus achievement and task allocation problems. The development of a swarm robotics system has resulted from the inspiration of various collective behavior systems from social animals examples in nature to their artificial design and implementation approaches (Brambilla, Ferrante, Birattari, & Dorigo, 2013).

In this work, the collective behaviors are the main target of my research with the focus on the learning-based approaches.

4.2 Collective Task Learning in Swarm Robotics

With the wide variety of problem descriptions and approaches in swarm robotics research, there is no universal and unified definition of swarm robotics system models and applications (Tan & Zheng, 2013). Most of the existing algorithms in swarm robotics need to deal with common features in a swarm, such as simple and scalable algorithms,

decentralization, local and limited communication/interaction, and parallelism. The most challenging part of such designs is the coordination/cooperation between agents while considering the behavior of each robot. Many research works discussed different approaches to simulate, model, or learn the collective behaviors in the swarm robotics system (Tan & Zheng, 2013; Brambilla et al., 2013; Bayındır, 2016). This section focuses on the learning approaches that can adapt the swarm behaviors based on the experiences, which I think more robust compared to other methods. Adaptation or flexibility is a robust phenomenon in the society of insects, and so these features are expected in the design of any swarm robotics system. Moreover, in some cases, we have no idea about how such behaviors can be formed. So it is too difficult to develop models or algorithms to mimic such behaviors from our nature.

Offline learning and online learning are two types of learning-based approaches (Bayındır, 2016). Usually, offline learning takes place at the design phase of the controllers in the swarm robotics. Evolution strategies such as genetic algorithms are a common approach. Some research works explored the uses of neural networks for robot controllers and then optimized by evolution algorithms, rule-based systems, or supervised learning. On the other hand, in online learning, the robots will adapt their behaviors during execution. Reinforcement learning and supervised learning are the most common online learning approaches for dynamically adjusting the behavior models/parameters from directly the received observations.

Evolution robotics, which introduced by Nolfi and Floreano in 2000 (cited in Brambilla et al. 2013), is the type of method that applies evolution algorithms to the robotics system. This method starts with defining the initial random seeds of individual

behaviors. In one iteration, some experiments are executed in which one unified behavior for all robots per experiment. For evaluating the performance of the swarm from each behavior, a fitness function is used. Next, a selection process with the genetics modifiers of cross-over or mutation will be executed to prepare the next generations for subsequent iterations. The individual behavior can be represented by finite state machines, virtual force functions, or recently artificial neuron networks. The drawbacks of evolution strategies are the computational expensive and not convergence guarantee for swarm robotics tasks.

Reinforcement Learning has an extended application history in the single-robot domain (Kaelbling et al., 1996; Sutton & Barto, 2018;). Recently, there are extension works of the multi-agent reinforcement learning (MARL) for multi-robot systems such as the review on cooperative MARL (Panait & Luke, 2005), a summary of recent advance of MARL for robotics (Wu, Xu, Wang, & He, 2011), and the discussion of deep reinforcement learning for swarm systems (Hüttenrauch, Adrian, & Neumann, 2019). The principle of applying reinforcement learning for swarm robotics is behavior adaption through the collected experiments and feedbacks (reward signals) when the robot agents are performing the actions in the environment. There are still many issues in applying multi-agent reinforcement learning algorithms for the multi-robot systems such as the credit assignment, dimension curve (high dimensional or continuous state-action spaces), partially observation states, and non-stationary (Rossi, Bandyopadhyay, Wolf, & Pavone, 2018). However, with the recent breakthroughs of deep reinforcement learning approaches, this kind of method shows promising future improvements in our collective task learning efforts.

The following subsections will discuss two widespread problems in the collective behaviors of swarm robotics. These discussions will give the literature backgrounds for our proposed methodologies in chapter 6.

4.2.1 Multi-robot Navigation

Navigation is an essential task in any swarm robotics system. A basic collective navigation scenario is each robot is assigned with a target location to navigate in an environment. There are many limitations of the agents in the swarm robotics, such as limited sensing, limited localization capabilities, and limited knowledge of the environments. Therefore, there are many developed variants of the above scenario with additional assumptions and conditions on the setup of robots and environments. In multi-robot searching problems such as find and rescue, foraging, etc., all robots have to find the unknown targets in bounded or open environments. Recall that swarm robotics is autonomous, self-organized, and decentralized, so the role of a centralized controller for all robots is not existed or very limited. Because there is no information about the target, the robots have to do random exploration, and often communicate with neighbor robots or interact with the environments in some ways to locate the targets. Explicit or implicit communication protocols between robot agents are critical here. Some approaches employ sharing knowledge between agents such as the estimated Euclidean distance to a target, “hop-count” (number of hops to reach a goal), pheromones (took the idea of ant colonies), landmarks provided by the environments (Bayındır, 2016). Some other approaches dedicate some robots, which already discovered or knew information about the targets, as the coordinators or landmarks to guide other agents.

Among the multi-robot navigation approaches, the communication-less ones are preferred because of their robustness, the benefits of reducing communication overhead, and low robot hardware configurations. Also, the communication should be given priority to other tasks, rather than the basic navigation task. The following subsections describe two significant research of such approaches: motion planning and learning-based strategies to develop a multi-robot collision-free navigation solution. In the basic scenario of these methods, the robots use their perception (typically through sensors) to avoid both static and dynamic obstacles (other agents) while moving to the target destination. During this process, the agents can collect different forms of feedback that give hints of the target location from the environment. Such hints could be relative distance, angle, reward, or penalty in a learning-based approach, etc. and vary depending on the solutions.

Motion planning. In this group of methods, there are centralized and decentralized approaches to determine the optimal trajectories of all robots. In the centralized approach, a central controller keeps track of the current position of all robots, solve the optimization problem to find the optimal navigation directions for all robots. The robots will receive such directions and proceed. This kind of approach is challenging to apply in practice because it heavily depends on the global positioning of all agents and the reliable communication between agents and the controller. There are some other drawbacks of this approach: the communication overhead, the difficulty of scaling to the increasing number of robot agents, which also make the optimization problem harder, and the risks of failed or lost communication agents.

In the decentralized approach, the robots decide their own navigation decision based on their perception. One of the most famous motion planning algorithms is the class

of obstacle velocity algorithms. The original idea of this algorithm is finding the velocity obstacles (VO), which are the set of robot velocities that can cause collisions with surrounding obstacles at a later time (Fiorini & Shiller, 1998). The authors introduced a heuristics tree search to find near-optimal trajectories that avoid the sets of robot velocities. In 2008 and 2011, two popular algorithms, named reciprocal velocity obstacles (RVO) (Van den Berg, Lin, & Manocha, 2008) and optimal reciprocal collision avoidance (ORCA) (Van Den Berg, Guy, Lin, & Manocha, 2011), were introduced that improved the previous algorithm by removing the assumptions of stability of other robots and adding a more efficient linear search.

Later, some variants of the RVO and ORCA algorithms were introduced such as the hybrid RVO (Snape, Berg, Guy, & Manocha, 2011) algorithm, the NH-ORCA (Alonso-Mora, Breitenmoser, Rufli, Beardsley, & Siegwart, 2013) algorithm for non-holonomic robots, the ORCA-DD (Snape, Van Den Berg, Guy, & Manocha, 2010) for robots with differentiate-drive constraints, etc. These algorithms are successful in multi-agent navigation and crowded simulation systems and require no communication between agents. However, these methods have some major drawbacks. The first drawback is the requirement or assumption of perfect observations of other robots' current position, velocities, shapes, and next motion plan. This assumption is impractical in the real case of robot deployment. Also, there are many settings in these algorithms that are not easy to tune. Therefore, some authors introduced the usage of a global positioning system to improve the localization accuracy of each agent (Bareiss & van den Berg, 2015). Besides, other authors suggested using communication protocol to transmit states and observations

between agents (Hennes, Claes, Meeussen, & Tuyls, 2012; Claes, Hennes, Tuyls, & Meeussen, 2012; Godoy, Karamouzas, Guy, & Gini, 2016).

Learning-based navigation. Supervised learning and reinforcement learning have been used in robotics for a long time. Some authors developed supervised learning methods to train an approximation function that maps directly from the input sensors to navigation commands (Muller, Ben, Cosatto, Flepp, & Cun, 2006; Zhang, Springenberg, Boedecker, & Burgard, 2016; Long, Liu, & Pan, 2017). These approaches lack generalization capabilities while requiring a lot of data to be collected and manually labeled. On the other hand, in 1992, reinforcement learning algorithms such as Q-learning and Temporal Difference (TD) algorithm were used to develop the obstacle avoidance navigation system for a single robot (Krose & Van Dam, 1992; Prescott & Mayhew, 1992). Later, there was less research on applying reinforcement learning for task learning of robotics, especially the case of swarm robotics because of some limitations of reinforcement learning in this domain. With the recent breakthroughs of deep reinforcement learning (see section 2.6 for more detail) in 2014, the interest of deep reinforcement learning on robotics has been revived. Some deep reinforcement learning approaches were proposed for single robot navigation problems (Tai, Paolo, & Liu, 2017; Kahn, Villaflor, Pong, Abbeel, & Levine, 2017; Zhu et al., 2017). Also, there were research works for multiple robot navigation tasks (Chen, Liu, Everett, & How, 2017; Chen, Everett, Liu, & How, 2017; Long et al., 2018; Ding, Li, Qian, & Chen, 2018).

4.2.2 Collective Foraging

Motivated by the capabilities of social animals (ants, bees, etc.) in discovering food sources using local communication, collective foraging tasks are also studied in swarm

robotics (Bayındır, 2016). In this task, the swarm agents need to explore the environment, find "*food*" items, and carry them to one or more designated areas, called "*nest*." Some real-world applications that are instance of this task are cleaning up or collecting toxicity chemicals or dangerous items, search and rescue, automatic construct or repair, and planet exploration, etc.

Formation and general approaches for the foraging problem: Cooperation is crucial for the success of the foraging task. With no information about the food sources or nests, communication is the only way for all robot agents to share and collectively achieve the objectives. In the survey, Bayındır (2016) pointed out three methods to design the communication in foraging tasks: a) shared memory, b) local modification of the environment, and c) direct communication.

Shared memory is a convenient way to share information between agents such as recent paths, nearby food sources/nests locations, etc. However, in many real-world applications, this approach is not practical because of the lack of scalability and the more complex design requirements of the robots. The second type of communication is inspired by the behavior of social animals, which is the production of pheromone to mark the trails. Similarly, different approaches mimic this behavior. The robots can put down some special "markers," or even some of them become the "markers" to direct or guide the others (Hecker, Letendre, Stolleis, Washington, & Moses, 2012; Hoff, Sagoff, Wood, & Nagpal, 2010). The third method of communication is the capabilities of robots that direct communicate and exchange information in a limited range. The transmitted information can be varied, such as food/nest locations, the relative position of nearby robots, the virtual pheromones or cardinality counts, etc.

Altering the environment approach has been used in many research. One of such research is the central-place foraging algorithm (CPFA), which was developed to model the ant's society (Hecker et al., 2012). This algorithm follows the altering environment method by allowing the robots to lay pheromone trails. The authors used a genetic algorithm to adapt different parameters such as the probability to remove a path, put down a pheromone, search again, etc. An extension of the CPFA algorithm was introduced with multiple homes to reduce the collision problem (Lu, Moses, & Hecker, 2016). Besides, Simonin et al. introduced another approach based on the artificial potential field to mark the direction to nests or food source (Simonin, Charpillet, & Thierry, 2014).

The direct communication approach has received a lot of attention in research works. A method of Goss and Deneubourg introduced the global transmitter that was used to transmit the home signal to all in-range robots (Goss & Deneubourg, 1992). For the out-of-range robots, they created a formation of a beacon system for an exploratory chain. Also, the more complex behavior-based (Goldberg & Mataric, 2000) algorithm was proposed by Golberg and Mataric in 2000. It required the robots to be equipped with bump sensors, five IR proximity sensors, a radio transmitter or receiver, and a positioning device based on ultrasound/radio triangulation. In 2010, Hoff et al. introduced the popular virtual pheromone and cardinality algorithms, which relied only on local communication. In these algorithms, the robots are in either of two roles: walker and beacon. While the walker robots explore, collect, and return food items, the beacon ones stay fixed at a location and communicate with neighbors to exchange the virtual pheromones or cardinalities numbers. This information is used to form dynamic paths to the food or nest. Hoff et al. use a heuristic function to decide dynamically the role of a robot based on the number of neighbors.

Learning-based approaches in foraging. Different evolution and machine learning algorithms have been used in multiple collective foraging approaches. Hecker and Moses (2012) exploit a genetic algorithm to find the optimal parameter in their CPFA algorithm. Nine parameters defined the different probability related to the agent decisions (such as searching, laying a pheromone, removing a trail, etc.) and selection values of pheromones. Their genetic algorithm started with the first generation of a randomly sampled for each parameter. Then, more generations were created by recombination and mutation techniques. To evaluate a generation, a fitness value, based on the total number of gathered resources, is used.

Similarly, another approach also used a genetic algorithm to optimize the selection of following a pheromone trail or return to the last found resources location (Letendre & Moses, 2013). In 2015, another genetic-based approach was used in a study of the effect of the resource distribution on different gathering strategies (Van Essche et al., 2015). The resources are distributed in two ways: one distributed them randomly on the entire area, and the other distributed them in a concentrated pile. For each resource distribution, they tested three different gathering strategies. In the first strategy called solitary, the robots randomly search for a resource and return without communication with others. The remaining two strategies are called “recruiter” and “recruitee”. While in the “recruiter” strategy, the robots randomly search the resource and inform other robots, the robots in the “recruitee” strategy will wait for the signal of resource location to pick up.

Reinforcement learning is also a popular approach for multi-robot foraging tasks. In 2003, a reinforcement learning approach was used to help the robots selecting their optimal foraging strategies (Ulam & Balch, 2003). In this approach, each robot can choose

one of three following plans: collecting red attractors only, collecting blue attractors only, and general foraging action (can collect any attractors). Also, a multi-Q learning algorithm was introduced for the multi-robot foraging problem (Guo & Meng, 2010). The author used a distributed approach to form a dynamic correlation matrix that stored both the Q-values of each robot and its neighbor. Although the novelty of this method, the biggest drawback is the heavy dependency on communication and state-sharing between agents. Recently, a study on the exploration-exploitation dilemma on the multi-robot foraging task was conducted (Yogeswaran & Ponnambalam, 2012). The traditional Q-learning algorithm was used along with different learning policies such as greedy, ϵ-greedy, simulated annealing, Boltzmann distribution, optimistic initial value (OVA), and probability matching. They concluded that, compared to other strategies for this task, the OVA policy is more practical and more efficient.

CHAPTER V – DEEP LEARNING FOR INFORMATION DIFFUSION ON SOCIAL NETWORKS

In this chapter, I motivate and introduce different deep learning approaches to model the information diffusion process, build a useful predictive model, and analyze the user influence function. Finally, I propose a user-level multi-feed weighted embeddings model to capture the user network interaction and topic diffusion on Twitter.

5.1 Information Diffusion Modeling and Volume Prediction on Twitter

Section 3.2 has summarized different methods for information diffusion modeling on social networks. While explanatory models, which includes epidemic models and influence models, try to capture the existing dynamic structure and behaviors of individual or nodes in the diffusion graphs, predictive models are specially designed to accurately forecast the information diffusion rate in the future (Li, Wang, Gao, & Zhang, 2017). As pointed out by Guille et al. (2013), these models only work with some underlying constraints such as explicit knowledge of network, or close world assumption (that the influence effect of external sources is not existing). Because of the high dynamic of the network structure of social networks, a successful model of information diffusion will depend on multiple factors. Some of the most important factors are the discussion topic, the user's network structure, sentiment, user location, and the existence of influential users (or topic leaders or content producers) that related to such topics.

There is multiple research in social networks that concentrated on the relationship between information flow and discussion topics. A framework, introduced in 2014, used linear multivariate Hawkes processes and the LDA (Blei, Ng, & Jordan, 2003) topic model for information diffusion modeling in social media (Pinto & Chahed, 2014). Besides the

topics and contents, the underlying network structure also has a crucial role in information propaganda on the social network. A previous study in 2003 discussed how network structure performs an essential role in information diffusion (Reagans & McEvily, 2003). Also, the authors concluded that network range and social structure are more important than the connection strength between two accounts for effective information diffusion, social capital, and knowledge transfer.

On social networks, users regularly express their opinions about a particular topic. Also, the rate of diffusion changes over time, depending on the dynamics of the mood/ sentiment of ongoing discussions. For example, a study showed the complicated dynamics between information diffusion and the sentiment of tweets (Ferrara & Yang, 2015). Firstly, the authors discussed the emotional impact on the transmission rate and content popularity, and later they addressed the relation between different type sentiments and their temporal evolution. However, they focus on only the tweets sentiment but do not discuss other essential aspects of the tweet, such as their related topics. Beside, Naveed et al. show that sad news spread much faster than good news initially, while good news always maintains a steady spreading rate (Naveed, Gottron, Kunegis, & Alhadi, 2011).

Identifying the correct sentiment of tweets is an extremely challenging problem. Tweets are usually short texts (140 characters limits), so it is difficult to analyze their sentiment. Also, different tweeting behaviors can be identified, such as “tweet," “retweet," ”reply," “direct message," and “like". Although there are several existing works on sentiment analysis of user tweets, most of the sentiment detection algorithms are designed to identify user opinions rather than user behaviors. For example, one study proposed a model that collects corpus from Twitter to performing opinion mining and sentiment

analysis (Pak & Paroubek, 2010). Besides, Thelwall et al. implemented the SentiStrength algorithm to determine the strength of sentiments from an informal text (Thelwall, Buckley, Paltoglou, Cai, & Kappas, 2010). One of the research on the relation between the sentiments and user behaviors is a content-based analysis of interestingness on Twitter using LDA and regression methods to show how tweets containing negative sentiment travel faster when compared to positive sentiment tweets (Naveed, Gottron, Kunegis, & Alhadi, 2011). The number of tweets and retweets may give useful hints on the popularity of topics. However, it is necessary to analyze the replies to understand the sentiment associated with the topic or hashtag. Moreover, the context and mood of other users are also essential for this purpose.

Contributions. Information diffusion is an active research area in which researcher studies how information is spread over social networks. Therefore, this section is mostly based on one of my very first research in this domain (Hatua, Nguyen, & Sung, 2017). We characterize the information flow using a multivariate time series model with three dimensions: volume, influence, and sentiment with ten different features. The reason to select these dimension and features are discussed in previous paragraphs. To evaluate our modeling method, we have collected 27.5 million of tweets in one month, performed data preprocessing, and finally developed our information diffusion dataset. After that, different temporal patterns of information diffusion features are revealed using time series clustering algorithms with Dynamic Time Warping (DTW) as the distance measure. Finally, we propose a prediction model of information diffusion features based on Long Short-Term Memory (LSTM) neural network architecture, which outperforms the traditional time series prediction model (ARIMA).

5.1.1 Related Work

In 2010, Yang and Leskovec introduced the Linear Influence Model (LIM), which proposed that the global information diffusion rate of a topic (volume) is the summary effect of all individual user influence effects. The most important part of their work is the representation of individual user influence function using a non-parametric fixed-length time series variable (Yang & Leskovec, 2010). This model can not only model the influence of each user but also predict the global diffusion volume in the future timesteps.

In 2012, Colbaugh and Glass proposed a derived stochastic hybrid dynamical systems (S-HDS) model to analyze and predict information diffusion processes under some realistic topologies (Colbaugh & Glass, 2012). Motivated by the importance of social influence, the authors performed formal stochastic reachability analysis. They concluded that the community structure and core-periphery structure are the two main factors that determine the outcome of social influence processes. Finally, an ensemble of decision trees approach was developed for predictive analysis and early warning of diffusion events. The authors also performed three case studies involving topic propagation, large-scale protests events, and cyberattacks to demonstrate the effectiveness of their early warning method.

Recently, a retweet-based information cascade approach to predict the information diffusion patterns on Twitter was introduced (Kafeza, Kanavos, Makris, & Vikatos, 2014). Based on the analysis of users' retweet behaviors, they identified various Tree-Shaped Tweet Cascades patterns. Later, they selected a set of four most popular patterns among them. Next, they used the set of user profile features, tree-shaped tweet cascade patterns, and linguistic features of the tweet, to build a diffusion pattern prediction model. The main drawbacks of their approach are the lack of discussion of the time dimension and the small

evaluation dataset to have a significant result (tweets data of some hours that related to a single hashtag were collected).

5.1.2 Methodology

In this work (Hatua, Nguyen, & Sung, 2017), we want to develop a generalized model of information diffusion on social networks that can capture different perspectives of diffusion topics such as volume, sentiment, and reachability. Also, such a model can predict those characteristics in the future. Such a model can help us understand the patterns of information diffusion processes in terms of volume, sentiment, and influenced users.

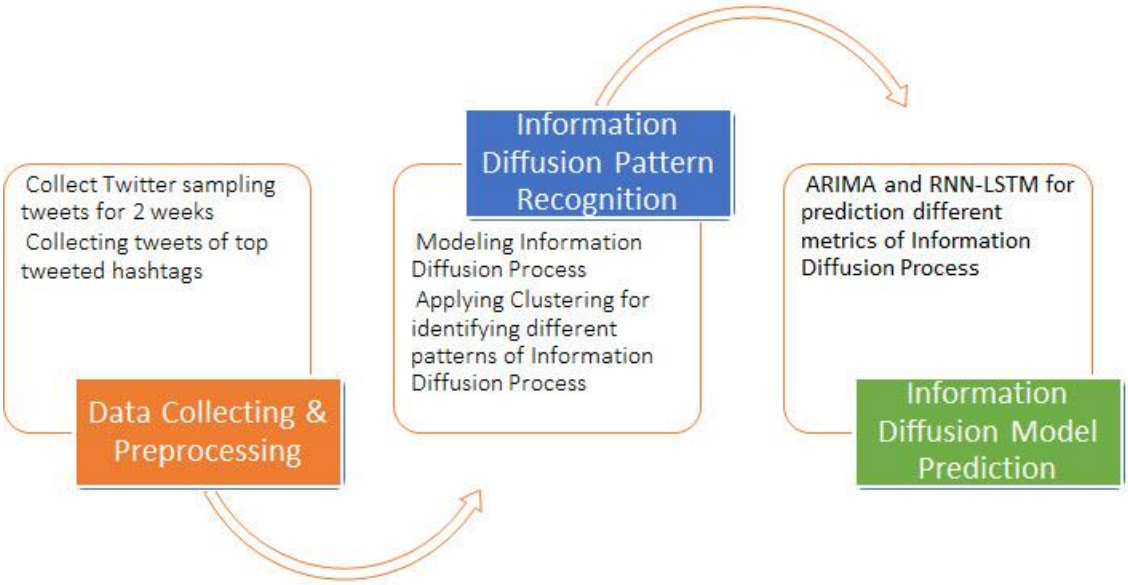


Figure 5.1 *The overview of our proposed information diffusion analysis framework which comprises different phases from data collecting, pattern recognition, and predicting*

As discussed in section 3.2 and the beginning of section 5.1, previous information diffusion methods have some limitations because of the dependency on network structure or close world assumptions. Therefore, our modeling approach is based on a non-parametric way. Regarding the experimental setup, Twitter is selected as the social network

platform because Twitter provides a very rich API to collet its public data. Figure 5.1 displays our general framework, which will be described in detail in the next subsection.

5.1.2.1 Information Diffusion Modeling

To avoid the dependency on the underlying explicit network structure, we represent different characteristics of information propagation cascades as time series variables in our model (Hatua et al., 2017). Therefore, our problem becomes a multi-variate time series analysis one. After considering different perspectives of information diffusion, we have selected ten features in three dimensions as described below:

A. **Volume**: This dimension represents the reachability of a diffusion topic (hashtag in Twitter) in terms of size over time
    a. #**tweet**: this feature describes the total number of tweets that are related to a corresponding topic in a measured time step.
    b. #**retweet**: this feature describes the total number of retweets/replies that are associated with a corresponding topic in a measured time step
B. **Influence**: This dimension represents the reachability of a diffusion topic in terms of user influence over time
    a. **#direct_influence_user**: this feature describes the total number of users and mentioned users who directly interact with the corresponding topic in a measured time step
    b. **#indirect_influence_user**: this feature represents the total number of friends and followers of all users and mentioned users who indirectly interact with the corresponding topic in a measured time step
C. **Sentiment**: This one represents the sentiment effect of a diffusion topic over time

a. **#positive_percentage**: this feature represents the percentage of positive sentiment posts (tweets) that related to a corresponding topic in a measured time step
b. **#neutral_percentage**: like the previous feature, this one represents the percentage of neutral sentiment posts (tweets)
c. **#negative_percentage**: like the previous feature, this one represents the percentage of negative sentiment posts (tweets)
d. **#positive_average_score**: this feature represents the average score of positive sentiment posts (tweets) that related to a corresponding topic in a measured timestep
e. **#neutral_average_score**: like the previous feature, this one represents the average score of neutral sentiment posts (tweets)
f. **#negative_average_score**: like the previous feature, this one represents the average score of negative sentiment posts (tweets)

Our proposed information diffusion modeling method based on multivariate time series can be quickly developed from the raw data stream collected from social network platforms. Also, with the availability of the dataset, machine learning methods such as unsupervised learning for pattern recognition and deep learning can be used for prediction. The next subsection will describe the technique to collect raw data and preprocess them to develop our information diffusion dataset for further analysis.

5.1.2.2 Information Diffusion Data Collecting and Preprocessing

As discussed in previous sections, topics are one of the essential factors in information diffusion processes. Different topics have their diffusion characteristics or patterns during their lifetime. In the case on Twitter, topics are the set of hashtags (memes)

that users often tag in their posts (tweets, retweets, or replies). Tagging the topics in tweets is a necessary action that not only helps users keep track of their communication information flow but also dynamically forms a community (a group of users who have similar exciting topics). The information flow starts when some users initiate some of their first posts to set up the discussion topic. These users are considered as early adopters (Rogers, 2010). Later, these users will influence their friends or followers on the topic, and some of them will become content creators for such topics. There is also a group of users who rarely author contents but keep interacting on the topic (such as retweet, like, etc.). The information flow status of a topic is denoted by the number of users communities interacting on that topic.

To collect data for information diffusion analysis, we start with a list of interesting topics and relevant hashtags. On Twitter, a topic can be represented by different hashtags. For each hashtag, we need to keep track of all the related tweets, retweets, or replies. Moreover, the users who interact with such hashtags will also be captured during the monitoring time. From the raw dataset, the multivariate time series dataset of information diffusion needs to be developed. We measure our time series features every hour. Except for the sentiment features, the calculation of other features in our model is very straightforward. In our model, we need to calculate the sentiment score of each tweet to classify it into positive, neutral, or negative sentiment. In our experiment, Python NLTK (Loper & Bird, 2002; Bird, Loper, & Klein, 2009) with WordNet (Miller, 1998; Wallace, 2007) corpora, was used to accomplish this task. Later, for each timestep (hourly), we collect the group of positive, neutral, and negative sentiment tweets. From these groups, the percentage and the average score of sentiment features can be calculated.

5.1.2.3 Information Diffusion Pattern Recognition

After the previous step of information diffusion modeling using multivariate time series with ten features, a study needs to be conducted to analyze several existing patterns in the dataset. To analyze and recognize such patterns, we employed many time series clustering algorithms to cluster hashtags into groups based on their similarity. As our information diffusion model contains ten features, we performed the clustering for each element separately. In this study (Hatua et al., 2017), various clustering algorithms such as partitional clustering, hierarchical clustering, and TADPole clustering with DTW distance (Petitjean, Ketterlin, & Gançarski, 2011), were used to identify various diffusion patterns and their forms. With the clusters that we got from the previous step, we can assign each group with a class label. Here, each class of information diffusion will have different temporal features in terms of volume, sentiment, and user influence. Now, a pattern classification model of information diffusion can be developed to classify a new topic based on their observation data.

A. Time-series clustering for information diffusion pattern analysis

DTW as the time series distance measure. Each feature (subset) of our multi-variate time series dataset is a sequence of data points that have been measured in equally spaced time steps (hourly). Pattern recognition in time series means capturing the regulate patterns or events in such datasets. One crucial task in this domain is developing an effective way to measure the similarity between two time-series sequences. Similar to the distance measuring method of two points in multi-dimension space in math, researchers also develop different distance measures for time series. Comparing to Euclidean distance, which cannot take into account the similarity of two time-series if they are not in the same

frequency, DTW has come one of the most popular distance measures in time series analysis. In the 1970s, DTW (Berndt & Clifford, 1994) was used for speech and word recognition from sound waves source. Dynamic programming is used in the DTW algorithm to compute the distance between two time-series segments, as displayed in Figure 5.2. Although DTW distance requires more computations compared to Euclidean distance, it still is an effective time series distance measure. In the DTW algorithm, the starting and ending points in the current window of two series must match with each other. However, DTW allows the flexible (time warping) of the middle positions, which are shown using the dashed blue lines in Figure 5.2.

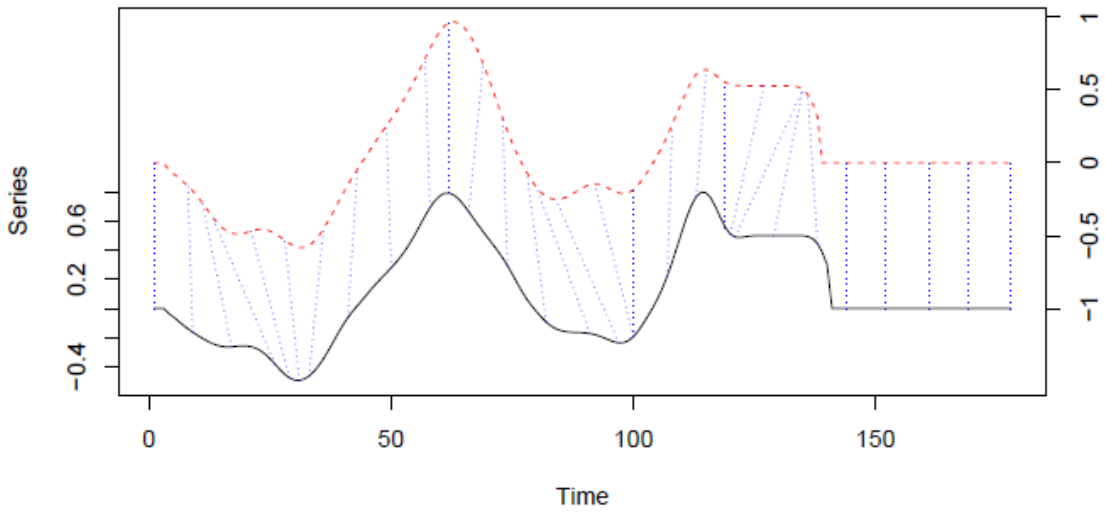


Figure 5.2 *A sample of alignment between two segments of time series that resulted from DTW distance measure (Sardá-Espinosa, 2017)*

Suppose $X =< x_1, x_2, \ldots x_n >$ and $Y =< y_1, y_2, \ldots, y_m >$ are two time series with n and m are their length respectively, the DTW uses the following steps to calculate the distance between X and Y:

Step 1: Local cost matrix (LCM). In this step, the distances between every pair of points from the two sequences are stored in a $n \times m$ matrix. The distance between two points i and j is calculated using the following equation: $lcm(i,j) = (\sum_v |x_i^v - y_i^v|)^{\frac{1}{p}}$, in which v is the dimension feature of each point, and p is the norm factor.

Step 2: Alignment and final distance. In DTW, an alignment path is defined as a set of mapping two data points from the two time-series sequences. Here, we always have two mappings $(0,0)$ and (n,m), and one point from a series can be mapped from multiple points from other series and vice versa. Dynamic programming approach will then be used to find the optimal alignment path $\phi = \{(0,0), \dots, (x_i, y_j), \dots, (n,m)\}$ that minimize the total local cost from every pair in such a path. Finally, the DTW distance can be calculated from the optimal path ϕ using the following equation:

$$DTW_p(X,Y) = \left(\sum \frac{m_\phi lcm(k)^v}{M_\phi} \right)^{\frac{1}{v}}, \forall k \in \phi \qquad (Eq. 5.1a)$$

In our work, DTW distance will be used in alignment with different clustering algorithms to recognize the patterns in our dataset. The next section will describe those algorithms.

Hierarchical clustering (Johnson, 1967): As the name suggests, this algorithm is a cluster analysis method that helps to build a hierarchy of clusters. Generally, there are two approaches to create such hierarchy clusters:

a) **Agglomerative**: a bottom-up approach in which each observation data point belongs to its cluster at the lowest level. Later, a pair of clusters will be merged in higher-level

b) **Divisive**: a top-down approach in which every observation data point belongs to the same group. Later, such clusters will be split recursively at the lower level.

The dendrogram (Friedman, Hastie, & Tibshirani, 2001) is often used to visualize the result of hierarchical clustering. It shows the tree of combining clusters from the bottom up or splitting groups from top-down while doing cluster analysis. Figure 5.3 displays a sample of a dendrogram.

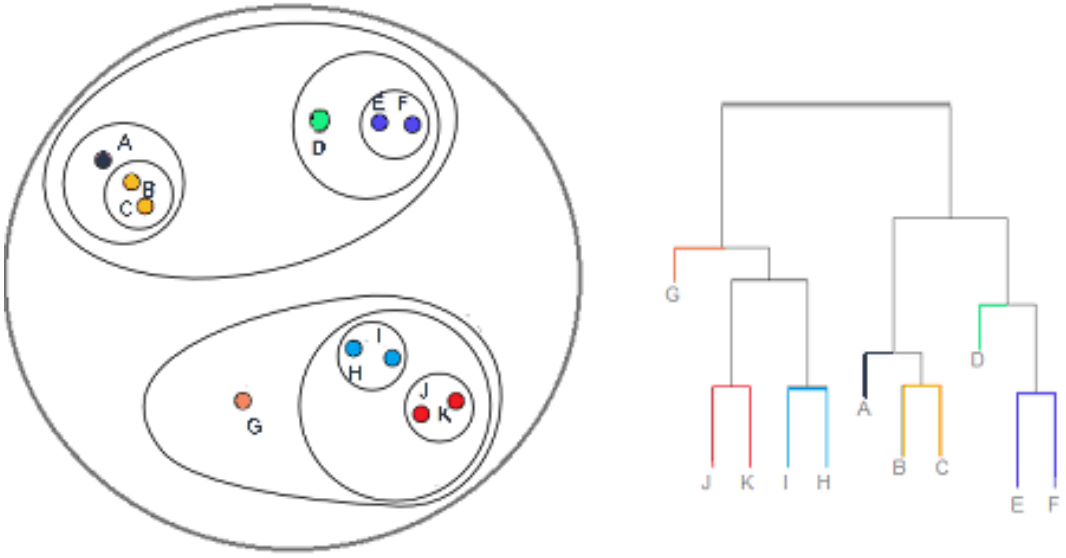


Figure 5.3 *A dendrogram sample of hierarchical cluster*

Partitional Clustering: This is a similarity-based clustering method, which requires the specified number of clusters at the beginning. K-means clustering (Hartigan & Wong, 1979) and k-medoids clustering (Rousseeuw & Kaufman, 1987) are the two most popular algorithms in partitional clustering. Initially, a fixed number of randomly data points (k) is chosen and assigned as centroids of such k clusters. Later, in subsequent iterations, the rest of data points are clustered into the groups based on similarity to the current k centroids. Before starting the next iteration, new centroids are calculated from the current status of clustering.

Fuzzy Clustering: In other clustering techniques, each data point of the dataset belongs to only one cluster at a time. However, the Fuzzy clustering technique allows each

data point of the dataset to be a member of a cluster with a particular degree (percentage). The sum of the membership degree (percentage) of a data point has to be equal to one. Then, a cluster membership matrix u of dimension $k \times N$ can be defined, in which k is the number of clusters, and N is the total of data points in the dataset. Each row in that matrix has the sum equal to one. Fuzzy c-means (Dunn, 1973) algorithm is a popular one in Fuzzy clustering methods. The fuzzy c-means algorithm starts by assigning randomly k-clusters membership degrees to each data point. Next, the centroids of such k clusters are computed by calculating the weighted mean across all data points (based on their membership degrees). The k-clusters membership degrees of all data points will be repeatedly updated so that the distance of each point to all clusters is minimized.

TADPole Clustering (Begum, Ulanova, Wang, & Keogh, 2015): This is a time-series data clustering method using DTW distance. Here, similar to the partitional clustering method, the centroid of a cluster is an element of the dataset. This method starts with finding the series in the dataset that has many close neighbors (compared in DTW distance space), using a predefined lower and upper bounds of the DTW distance. A cutoff value of distance has to be preset to determine if two series are neighbors. Also, this method introduces a DTW calculation pruning approach to make the clustering steps faster.

Cluster Evaluation. Performance evaluation is essential to quantify the effectiveness of the clustering results because clustering is unsupervised. Different cluster evaluation metrics, called cluster validity indices (CVIs) (Arbelaitz, Gurrutxaga, Muguerza, PéRez, & Perona, 2013), have been proposed. In this work, we selected some popular indices among them. Internal and external are types of two cluster validity indices. While internal indices do concern how the clustering (partitioning) is happening internally,

the external one does not. An example of internal CVI is the Silhouette index, while an example of external CVI is the Variation of Information (Meilă, 2003) index. In our experiments, we did not use external CVIs. The following internal CVIs are used: Silhouette, Dunn, COP, Davies-Bouldin (Kim & Ramakrishna, 2005), Davies-Bouldin, Calinski-Harabasz, and Score Function (Saitta, Raphael, & Smith, 2007). Table 5.1 presents the descriptions and optimization conditions of the internal CVIs used in our experiments.

Table 5.1 *List of internal CVIs and their optimization conditions that are used in our experiments*

Index	Name	Descriptions
Sil	Silhouette	Maximized to get better clustering results
D	Dunn	Maximized to get better clustering results
COP	COP	Minimized to get better clustering results
DB	Davies-Bouldin	Minimized to get better clustering results
DBstar	Modified Davies-Bouldin	Minimized to get better clustering results
CH	Calinski-Harabasz	Maximized to get better clustering results
SF	Score Function	Maximized to get better clustering results

B. Information diffusion pattern classification

The previous time-series clustering approach helps to discover different temporal patterns of information transmission in our dataset. The next question is how to know what is the diffusion pattern of a new hashtag (not exist in the dataset), and to predict information diffusion characteristics in the future timesteps. This section explains our method of applying *k-NN* to answer the first question above. It is observed that after the previous step, we have different clusters of data points. These clusters can be used to construct a classification model to identify the patterns of information dissemination of a new hashtag.

The simple idea is to find which cluster the temporal data of a new hashtag belongs. The steps to classify such patterns of a new hashtag as described below:

- **Step 1**: Collect raw data and develop a time series dataset for the new hashtag that needs to find the diffusion patterns (by following our modeling method)

- **Step 2**: Use DTW distance to find *k* closest points from each of the clusters to the new time series dataset

- **Step 3**: Find the mean of those selected *k* points of each cluster

- **Step 4**: Compare the DTW distance between the new data point and the above *k* mean data point (can be considered as the distance of the new data point to each cluster)

- **Step 5**: Assign the new data point to the cluster that has the closest mean to the new data point. The new hashtag is expected to have similar diffusion patterns of the found cluster

5.1.2.4 Information Diffusion Prediction

Predicting the future characteristics of the information diffusion process is an essential and fundamental task. As our information diffusion model is based on multivariate time series, a forecasting method for this kind of data has to be used. Time series analysis has a long history in many disciplines, such as commercial/sales forecasting, budgetary analysis, stock market analysis, inventory studies, etc. These problems are too hard for any parametric model to work in real life. Therefore, traditional time series analysis tries to identify different components such as trend, seasonal, and noise (Mills, 1991). Time series analysis makes use of some characteristics of time series, which are stationarity, differencing, and auto-correlations. Different time series models for curve fitting regression have been developed, such as Autoregression, Moving Average, or

Autoregressive Integrated Moving Average (ARIMA), etc. Recently, stateful RNNs based models have been applied successfully in sequence data, which also includes time series.

Autoregressive Integrated Moving Average (ARIMA). ARIMA method, which also called the Box-Jenkins approach, has been introduced from 1970 for time series forecasting (Box, Jenkins, Reinsel, & Ljung, 2015; Makridakis & Hibon, 1997). ARIMA models are typically applied to non-stationary data with a few of differencing steps to make it stationary. In statistics, a time series variable is stable if its mean value does not change over an amount of time. Next, differencing is the process of calculating the difference between the current data values with their previous ones. ARIMA generalizes the Autoregressive Moving Average (ARMA) by introducing the Integration step (differencing process). The three different aspects of ARIMA models (Asteriou & Hall, 2011) are described as the following:

- AR: Autoregressive. This component indicates that the interest variable is regressed using its prior values.

- I: **Integrated**. This component indicates the usage of differencing steps to make the interest variable stationary.

- MA: **Moving Average**. This component is introduced to deal with the regression errors that occur during the curve fitting. The model uses a linear combination of error terms to represent this component.

An ARIMA (Box, Jenkins, Reinsel, & Ljung, 2015) model has three parameters, naming p, d, q. The p component indicates the number of autoregressive terms (how many past values to be used or also called lag order). The d components denote the total of non-seasonal differencing steps that are needed for making the interest variable stationary.

Lastly, the q component represents the size of the moving average window. The forecasting value Y of the interest variable is described by the following equation:

$$(1 - \sum_{i=1}^{p} \phi_i L^i)\,(1 - L)^d Y_t = (1 + \sum_{i=1}^{q} \theta_i L^i)\epsilon_t \qquad (Eq. 5.1b)$$

In the above equation, L represents the lag operator that returns the i^{th} previous value, $\{\phi_i,\ 1 \le i \le p\}$ is the set of autoregressive parameters, and $\{\theta_i, 1 \le i \le p\}$ is similarly the set of moving average parameters. Besides, the d parameter denotes the order of differencing to make the targeted time series variable stationary. To deal with the autocorrelated errors at different timesteps (ϵ_t), some number of moving average terms ($q \ge 1$) are introduced in the final ARIMA models.

Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) is an advanced type of recurrent neural network, in which the input of the current time step is taken from the output of the previous time step. LSTM was designed by Hochreiter & Schmidhuber, which tackle the long-term dependencies problem of RNN. Such the problem is RNN cannot retain the long-term dependencies data but only the recent information. Therefore, in a long-running time, RNN does not provide efficient performance. However, LSTM can maintain the data for a long time. Some of the most popular applications of LSTM are machine translation, language modeling, image captioning, handwriting generation, and question answering chatbots. Especially, LSTM is very efficient in processing, classifying, and predicting time series data. LSTM uses a chain structure containing four gates and some memory cells (see Figure 2.8 for more details). Information is maintained by the cells, and the gates do the memory manipulations. There are three gates: forget gate, input gate, and output gate. In this thesis, LSTM is used to build

the prediction model of our information diffusion. The performance (discussed in the next section) shows the novelty of this method because of the availability of large dataset can help deep learning approximates better the complex actual model of the problem.

5.1.3 Experimental Work

5.1.3.1 Information Diffusion Dataset

To collect the data for our experiment, we need to have a list of interesting topics (hashtags on Twitter). However, we did not have any preference for such a list in hand. Therefore, we decided to capture the Twitter sampling data from July 1st, 2017 to July 14th, 2017, using Tweepy (Roesslein, 2009) Python library[3]. From this raw data of two weeks, we can collect all the mentioned hashtags with their volume size. There are approximately 1 million different hashtags in this raw data, and because we do not want to keep that much. Therefore, using a threshold of 200, we can discard most of the hashtags which do not have a large enough volume size of tweets. As a result, we have a list of 1,686 hashtags as the seeds for our information diffusion dataset. Later, from Jul 15th, 2017 to Aug 4th, 2017, we started using Tweepy Search API to gather tweets that were tagged with our list of hashtags, as mentioned above. Finally, about 27.5 million tweets were collected, followed by data preprocessing steps (see section 5.1.2.2 for more detail), to build the multivariate time series dataset of our proposed information diffusion model.

After performing data preprocessing, in our information diffusion dataset, we got ten time-series subsets which are corresponding to three dimensions with ten features. In our information diffusion time-series dataset, each subset has 1,687 samples (hashtags) with 467 measured time steps (hourly).

[3] Tweepy: visit http://www.tweepy.org/ for more detail

5.1.3.2 Information Diffusion Patterns Recognition

The first step of our experiment is performing unsupervised clustering to split our time series dataset into different clusters. One important decision we have to make is the initial number of clusters because this parameter will decide whether or not satisfactory results can be obtained. Because there are no clues about how many patterns in our dataset, we tried the number of clusters from four to ten. As discussed in section 5.1.2 mentioned above, the following interval CVIs were utilized to assess the performance of our clusters: Sil, D, COP, DB, DBstar, CH, and SF. As the feature variables in our time series dataset are considered independent, we will perform the clustering on each feature and keep track of their CVIs performance. While performing the experiments, the best number of clusters that have the best overall ranking performance among its CVIs indices will be recorded. Later, the best CVIs indexes values for each feature will be recorded in Table 5.2 below. In the table, while the first column denotes the name of each feature in our time series model of information diffusion, the second column displays the best number of clusters for the corresponding feature. The following columns in that table display the values of the CVIs in our experiment. Moreover, we also plot the merging curve of all observations that belong to each cluster in Figure 5.3 to Figure 5.12.

Clustering results of #tweet and #retweet volume: As shown in Table 5.2, for #tweet and #retweet features, the best number of clusters is six. Table A.1 and A.2 of Appendix A describe the CVIs comparison of the two above features. Also, the visualization of those pattern clusters is displayed in Figure 5.4 and 5.5, correspondingly. In these figures, the x-axis represents the time steps (hourly), and the y-axis represents the volume of tweets or retweets.

Table 5.2 *The best CVIs for each feature in our model*

Note: The character '-' in column D represents the invalid values

	CVIs	Sil	SF	CH	DB	DBstar	D	COP
#tweet	6	0.068	0.011	193.24	1.880	2.370	-	8.680
#retweet	6	-0.095	0.0017	55.75	1.505	1.887	0.002	0.899
#direct_ influence	4	0.130	0.020	286.13	1.470	1.530	0.001	0.900
#indirect_ influence	4	0.070	0.020	214.94	1.110	1.380	$1.1e^{-16}$	0.400
#negative_ percentage	6	0.621	0.0316	389.82	1.045	1.318	-	0.323
#neutral_ percentage	6	0.020	$7.4e^{-9}$	$1.01e^{3}$	1.390	1.860	-	0.360
#positive_ percentage	6	6.890	7343.5	255.73	1.387	1.628	-	3.510
#negative_ average_score	6	0.867	0.0673	250.42	1.834	2.166	0.038	0.933
#neutral_ average_score	6	0.167	$5.8e^{-11}$	752.13	5.110	5.550	-	0.360
#positive_ avg_score	6	0.040	0.020	540.15	1.758	1.987	-	0.061

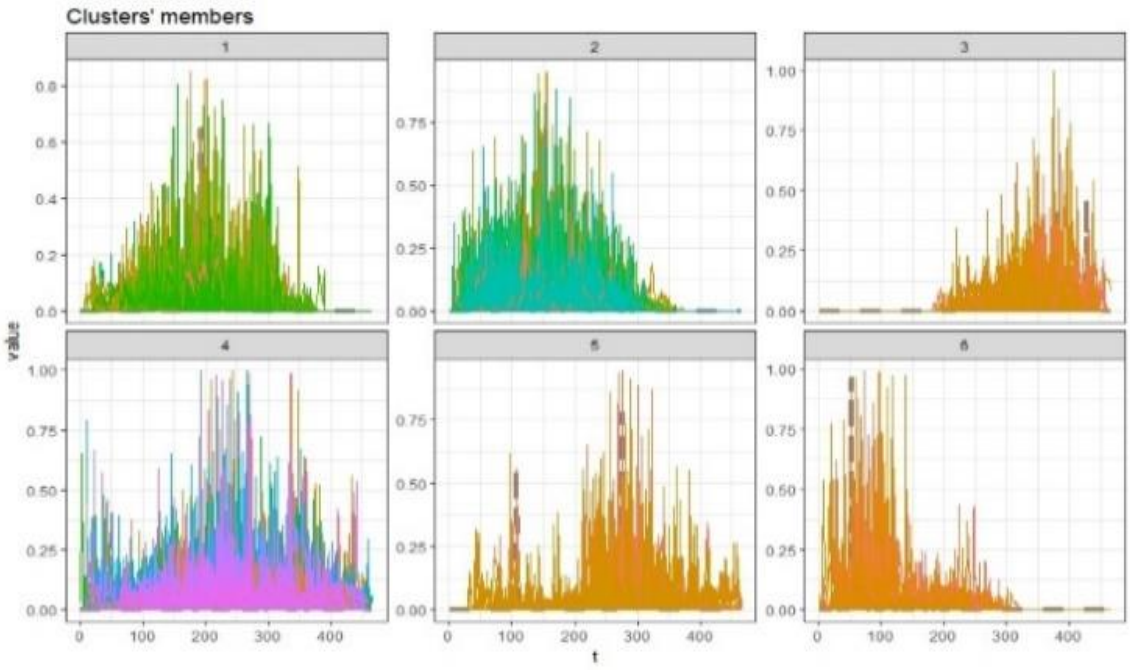


Figure 5.4 *The visualization of six clusters of #tweet feature*

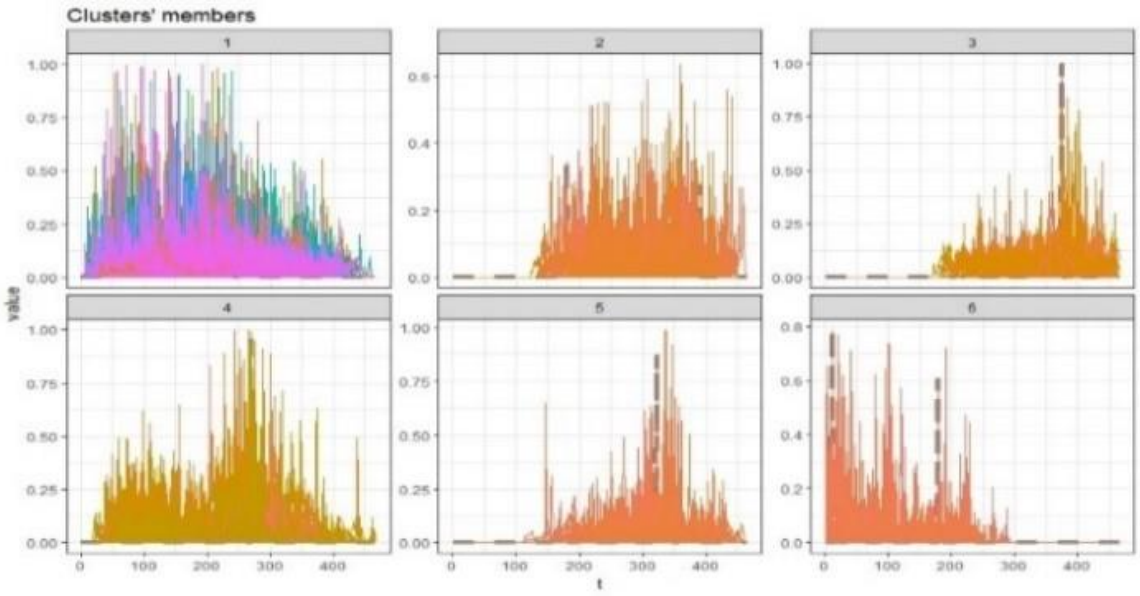


Figure 5.5 *The visualization of six clusters of #retweet feature*

Clustering results of #direct and #indirect influence: As shown in Table 5.2, for #direct and #indirect influence features, the best number of clusters is four. Table A.3 and A.4 of Appendix A present the CVIs comparison of these two features. Besides, the visualization of those pattern clusters of these two features is also displayed in Figure 5.6 and 5.7, correspondingly. In these figures, each subplot contains the drawing of all time-series of related hashtags in the same cluster.

Clustering results of the percentage and average score of #positive, #neutral, and #negative sentiment: As shown in Table 5.2, for sentiment features, the best number of clusters is six. Table A.5 to Table A.10 of Appendix A presents the CVIs comparison of these features. Also, the visualization of those pattern clusters of these two features is displayed in Figure 5.8 to 5.13, respectively.

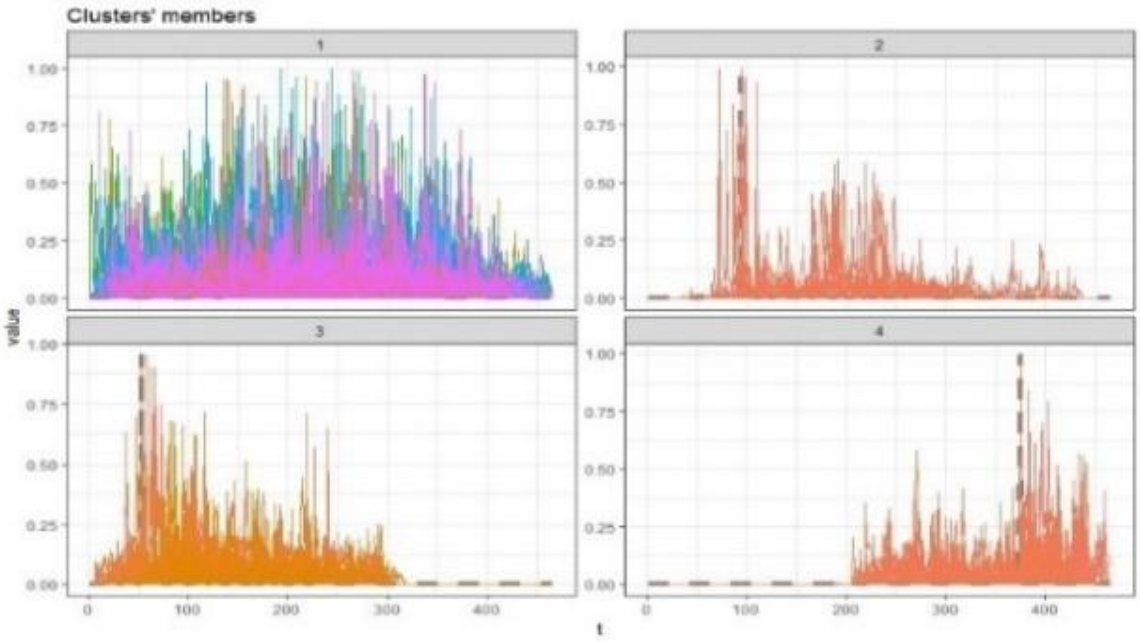


Figure 5.6 *The visualization of four clusters of #direct_influence feature*

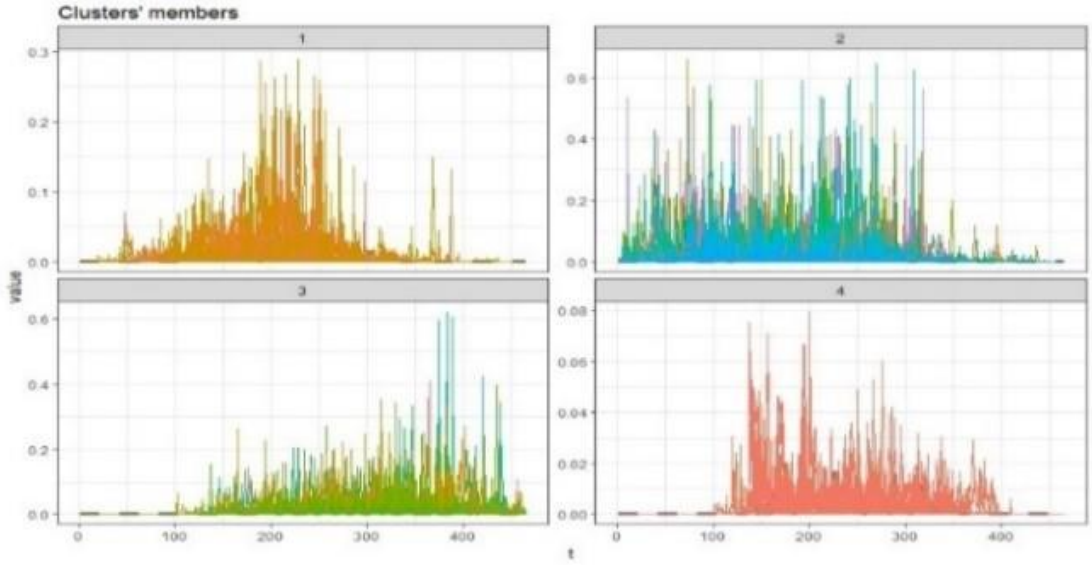


Figure 5.7 *The visualization of four clusters of #indirect_influence feature*

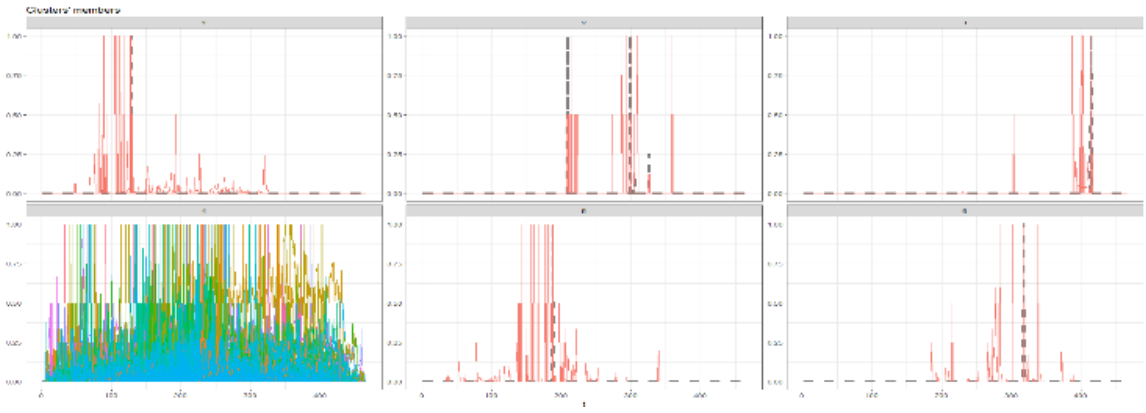


Figure 5.8 *The visualizations of six clusters of #positive_percentage feature*

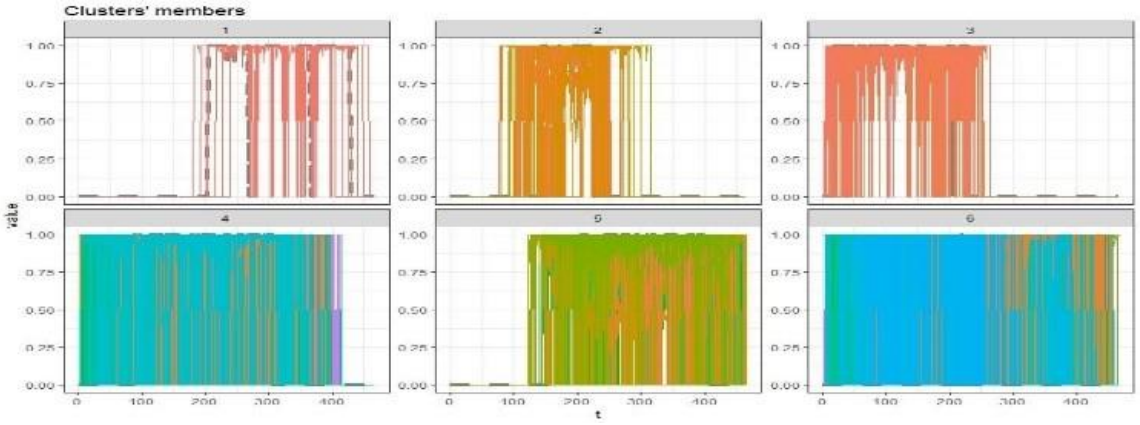


Figure 5.9 *The visualizations of six clusters of #neutral_percentage feature*

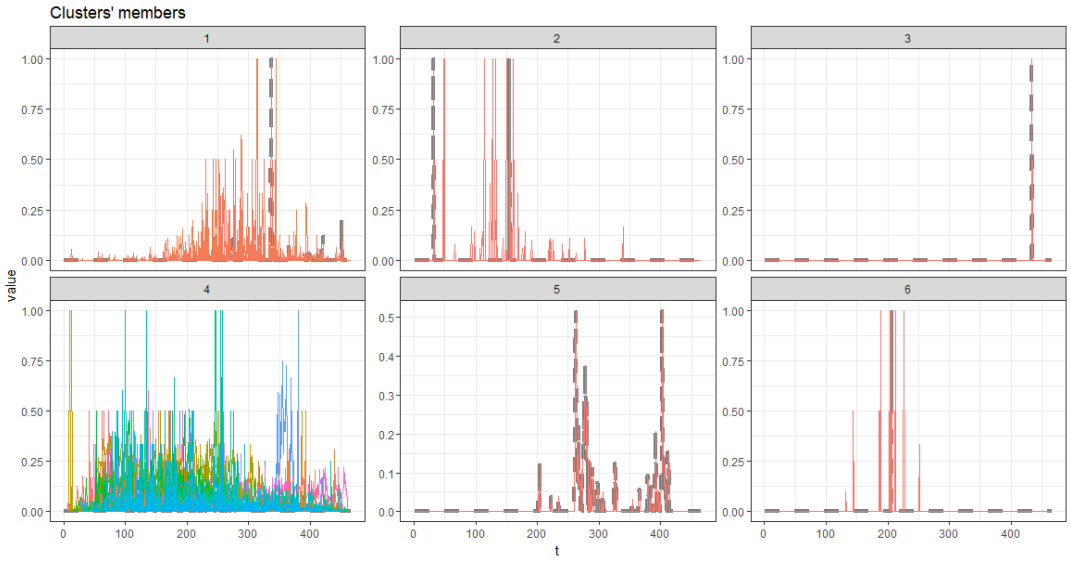


Figure 5.10 *The visualizations of six clusters of #negative_percentage feature*

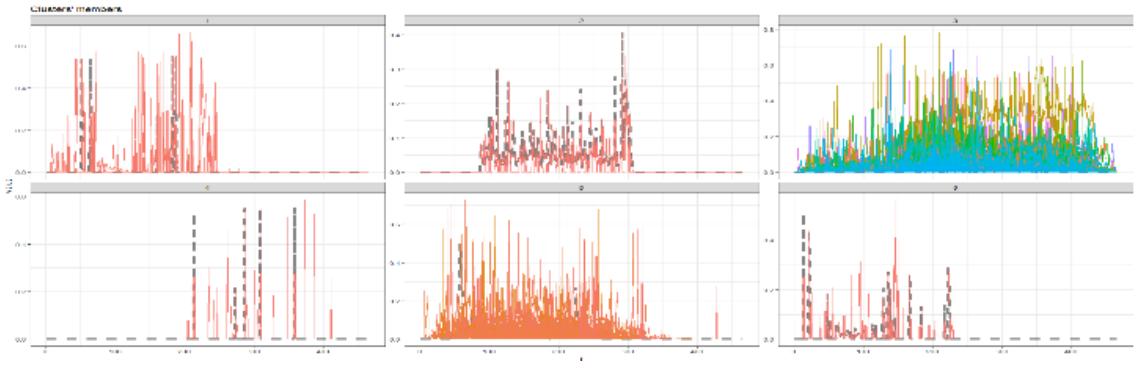


Figure 5.11 *The visualizations of six clusters of #positive_average_score feature*

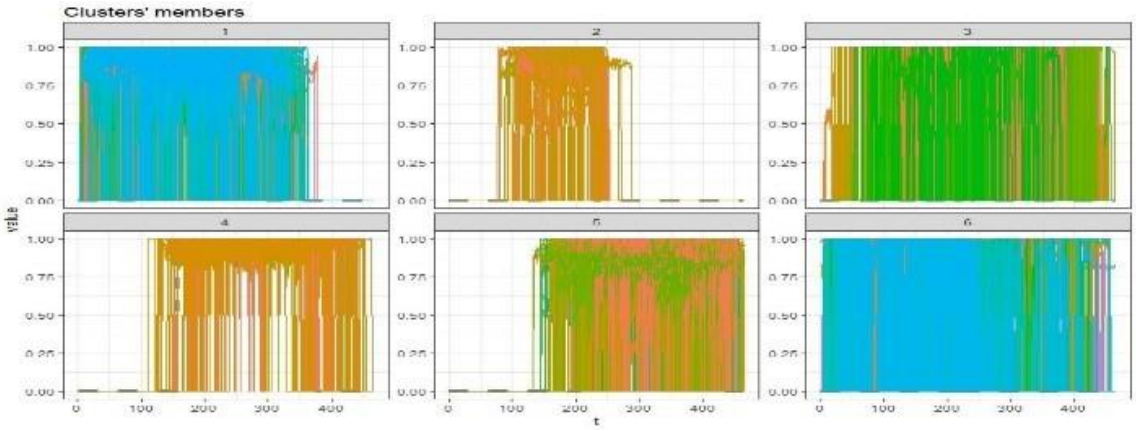


Figure 5.12 *The visualizations of six clusters of #neutral_average_score feature*

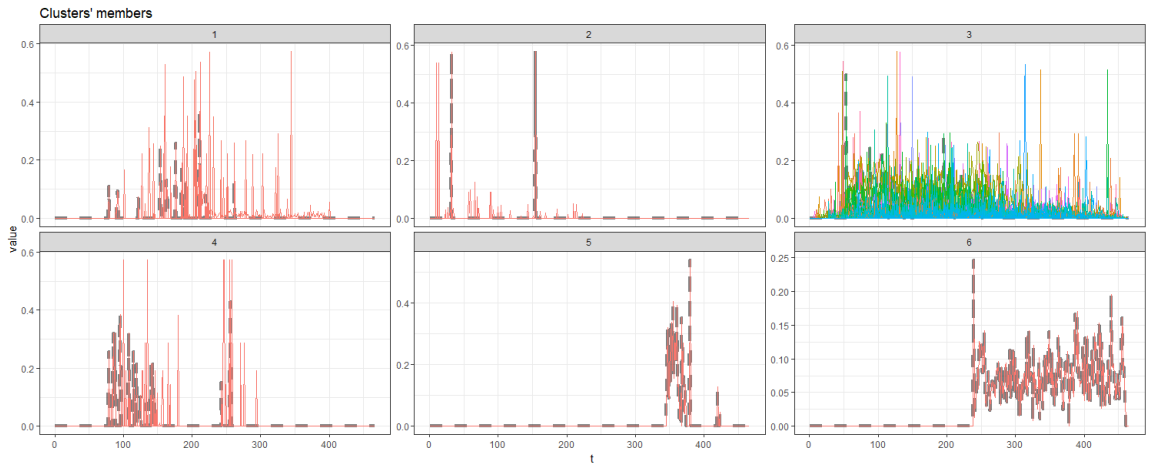


Figure 5.13 *The visualizations of six clusters of #negative_average_score feature*

Discussions of Information Diffusion Pattern Recognition. The clustering results and the displays of those clusters have demonstrated the typical patterns of

information propagation processes on Twitter. It can also be understood that the information flow of a topic on the Twitter platform shows some characteristics of time series such as trends, seasonal, or autocorrelated. Through our studying and analysis of those clusters, we can spot some of those common patterns that already been found before, such as the independent study of sentiment effect on information diffusion by Ferrara and Yang (Ferrara & Yang, 2015).

A. **Symmetric**. This kind of pattern displays the trend that the value grows slowly over time, reaches the peak, and slowly goes down. Cluster 1 in Figure 5.4 and cluster 1 in Figure 5.7 show this kind of pattern.

B. **Trending**. This pattern is typically displayed by the very popular/trending topics (hashtags), in which the values of the feature are consistently high. This pattern can be seen in cluster 4 in Figure 5.4, cluster 1 in Figure 5.5, cluster 1 in Figure 5.6, and some clusters in neutral sentiment graphs.

C. **Unexpected**. The topics, which belong to this pattern, show a high diffusion rate in their early stage, and gradually decrease the value over time. Cluster 2 in Figure 5.4, cluster 6 in Figure 5.5, cluster 3 in Figure 5.6, cluster 1 in Figure 5.8, etc. show this kind of pattern. The example hashtags of this kind of pattern are the ones related to natural disasters, earthquakes, terrorist events, etc.

D. **Anticipatory**. Some patterns are demonstrating low values in the early time and increasing at a later time. An example of this pattern is a football game. Before the game start, supporters begin to discuss it on Twitter and reach the highest peak in the day the game happens and drops rapidly right after the game.

E. **Spike / Transients**. Some patterns show suddenly high values in a short time.

Besides those patterns that can easily be spotted on the visualizations, some different patterns have not been discovered before. That demonstrates the promise of unsupervised learning when dealing with enormous data. As mentioned in section 5.1.2.3, these information diffusion patterns can be named as class labels to build a pattern classification model of information diffusion later. In case a new hashtag is introduced, by snapshotting its temporal time series data according to our model, we can quickly determine its propagation pattern by test its data on our classification model. That can give us an overall or brief idea of how this new topic will be evolved in social networks in the future. However, if we want to forecast its real value, the prediction model needs to be used. Such models are introduced in the next section.

5.1.3.3 Information Diffusion Prediction

As described in section 5.1.2, we employed ARIMA and LSTM models to forecast the future values of various information dissemination features. For the performance comparison of the two models, we applied a 70-30 train-test data splitting scheme, in which we train the models using the first 70% total time steps to, and assess the prediction performance of those models on the rest 30%. Also, we used the Root Mean Square Error (RMSE) to compare the performance of such two models.

ARIMA. For each feature variable of our dataset, we apply a grid search on the first 70% of the dataset to find the ARIMA (p, q, d) model parameters for each hashtag, correspondingly. To find the best prediction models for ARIMA, we used the '*Arima'* method in the '*forecast'* package of R[4]. Then, the founded models are used to forecast the

[4] Arima: check the documentation at https://www.rdocumentation.org/packages/forecast/versions/8.12/topics/Arima

values of the rest 30% of the dataset. The total prediction error in RMSE is calculated as the sum of RMSE of all predictions vs. the ground truth values of all hashtags.

Table 5.3 *RMSE performance comparison between ARIMA and LSTM on the prediction of our multivariate time series dataset of information diffusion*

Features	**ARIMA**	**LSTM 24 × 128**
#tweet	2.08	0.0088
#retweet	2.08	0.0085
#direct_influence	2.04	0.0036
#indirect_influence	3.09	0.0037
#positive_percentage	0.893	0.0154
#neutral_percentage	3.48	0.0087
#negative_percentage	0.148	0.0023
#positive_average_score	0.92	0.0101
#neutral_average_score	3.26	0.0095
#negative_average_score	1.91	0.0132

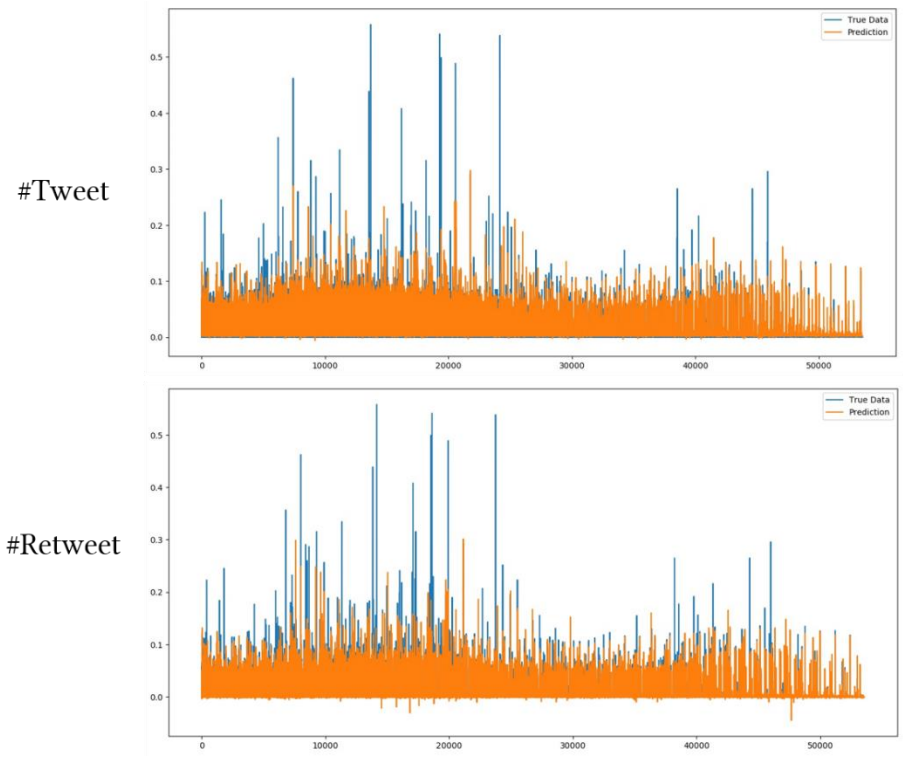


Figure 5.14 *RMSE performance comparison of prediction on the volume features*

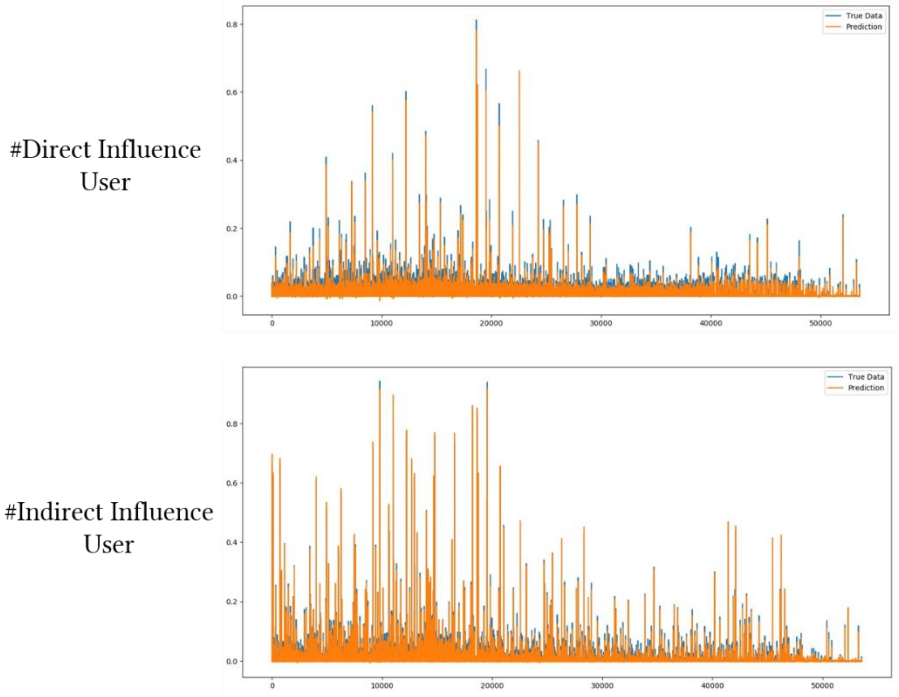


Figure 5.15 *RMSE performance comparison of prediction on the influence features*

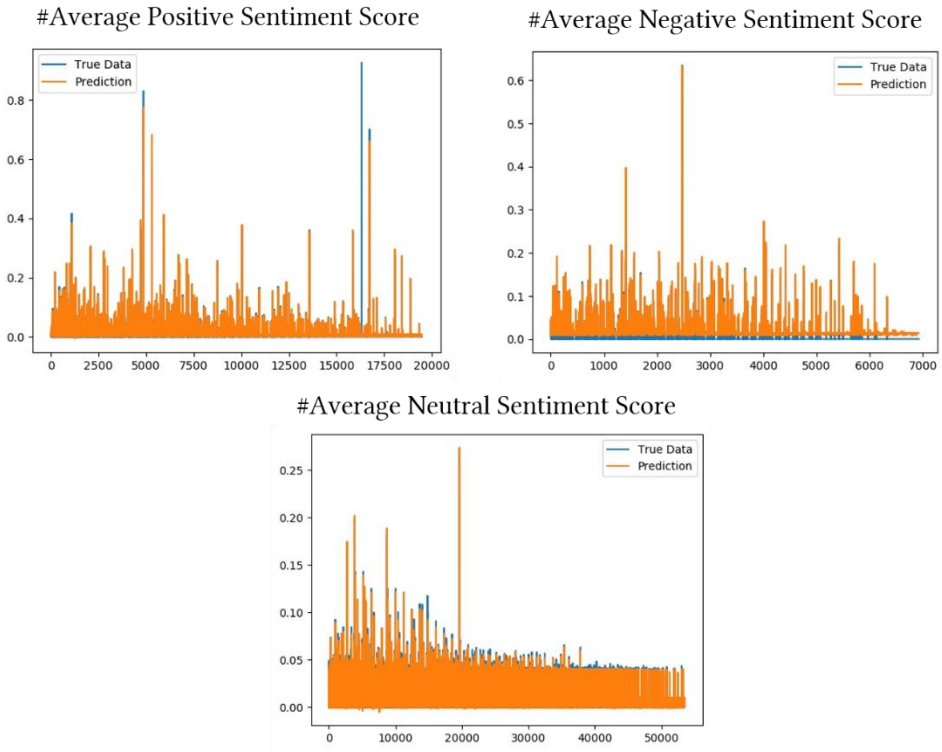


Figure 5.16 *RMSE performance comparison of prediction on the average score of sentiment*

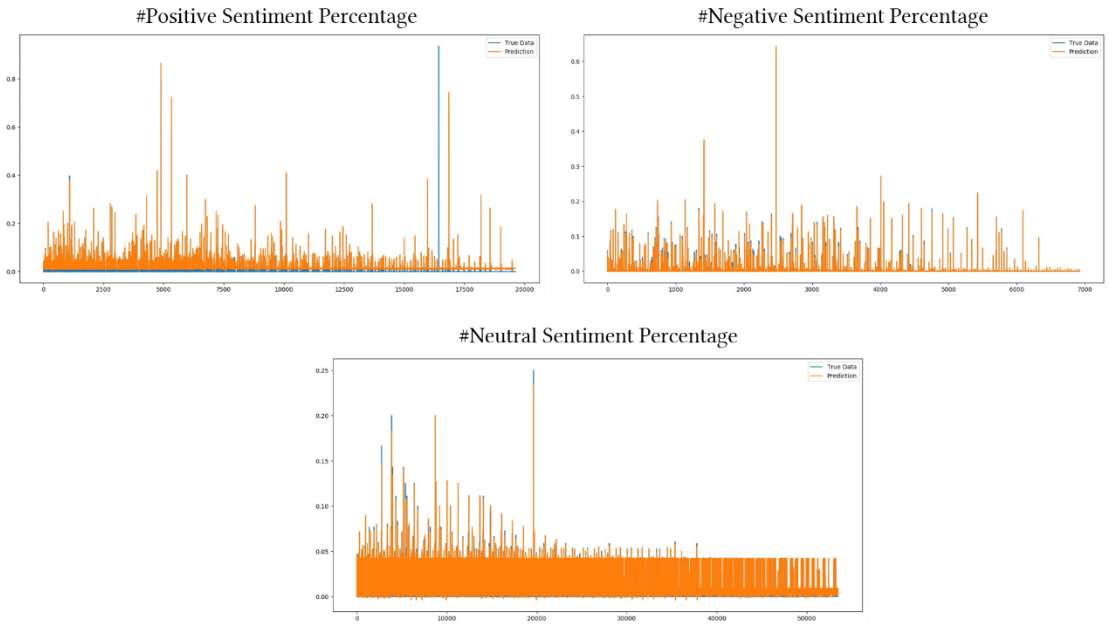


Figure 5.17 *RSME performance comparison of prediction on the percentage of sentiment*

LSTM. The architecture of the LSTM model that was used in our experiments has two layers. These two layers contain 24 memory cells and 128 memory cells, correspondingly. Furthermore, the window size of 24 was used, which means 24 previous timesteps values were used to predict the amount at the current timestep. We used 100 epochs to train all LSTM models. Figure 5.14 to Figure 5.17 displays the comparison of the forecast values of LSTM models and the ground truth values that are corresponding to our multivariate information diffusion models. Moreover, Table 5.3 presents the performance comparison of various ARIMA and LSTM models that were employed to forecast our multidimensional time-series dataset of information diffusion on Twitter. The performance comparison in Table 5.3 shows that LSTM prediction models outperform traditional ARIMA models with statistical significance difference. Furthermore, compared to other models, building LSTM models for each cluster of time series (using the pattern recognition results) can help to attain better performance.

This section introduced the approach of information diffusion modeling using multivariate time series, recognize different temporal patterns of those processes, and propose deep learning models to predict the future status of such processes. The source code and the dataset can be obtained from our public repository[5]. The next section will focus on the influence analysis of information diffusion.

5.2 Influence Modeling and Volume Prediction using Tweeting Behavior on Twitter

As discussed in section 3.2, the information diffusion processes depend on the content (popularity of topics, users' tweets, etc.), network structure, sentiment, and influence of related users, etc. For a topic, while it is challenging to keep track of the highly dynamic of underlying network structure, influence analysis of information diffusion becomes a hot research topic. Here, influence analysis includes influence modeling and influence maximization. Guille et al. (2013) have summarized different graph-based methods to model the influence/activation parameters between user nodes such as Linear Threshold (LT), Independent Cascade (IC), AsLT, AsIC, NETINF, NETRATE, INFOPATH, etc. There are also non-graph-based methods to model the influence on social networks such as the SIS model of Leskovec et al., the LIM model of Yang and Leskovec, PDE of Wang et al., etc. These models try to model the information diffusion rate by using a global influence probability (NETINF, NETRATE, etc.), or user-specific influence threshold (SIR, SIS, etc.), or a fixed-length window of individual influence function (LIM). However, the influence aspect of the information diffusion process shows very complex behaviors that these models are found difficult to capture in real-world applications. Motivated by the work of Yang and Leskovec, in this study, deep learning approaches have

[5] https://github.com/amartyahatua/informationdiffusion

been used to study and model the individual user influence function on social networks based on the historical data of user' tweeting behavior. Later, these models are used to develop a new volume prediction model.

Contributions. We proposed a new influence modeling approach and an influence-based volume prediction model in our research (Nguyen, Hatua, Pothuraju, & Sung, 2018). Our proposed influence modeling approach can help quantify the individual influence of each user based on their tweet and retweet behaviors and then develop prediction models of information diffusion processes based on the influence analysis results. Our main contribution is a set of new prediction models to forecast the number of nodes that will be influenced by other nodes in the implicit network associated with a specific topic. The performance evaluation shows that our proposed influence model outperforms the baseline Linear Influence Model in the accuracy metric for modeling the global influences of nodes over time and predicting the temporal volume of information diffusion processes. Additionally, an enhanced method to measure the sentiment of tweets accurately using a new data handling technique in conjunction with various machine learning algorithms was also proposed. The details of our experiments and their performance are discussed in the next subsections.

5.2.1 Related Work

In 2010, Yang and Leskovec introduced the Linear Influence Model (LIM), which proposes that the global information diffusion rate of a topic (volume) is the summary effect of all individual user influence effects. The most important part of their work is the representation of individual user influence function using a non-parametric fixed-length time series variable. Figure 5.18 displays the examples of three influence function I_u, I_v, I_w

of three correspondent users u, v, w. In this example, these three influence function I_u, I_v, I_w have the same length, but they have different starting timesteps, which are t_u, t_v, t_w. The LIM model tries to predict the temporal aspect of topic diffusion, denoted by the volume variable, through the diffusion rate (individual influence function) of all user nodes that take part in the diffusion process. If $V(t)$ is defined as the global diffusion volume of a topic at timestep t, and $A(t)$ is the set of active users that are interacting with such a topic before timestep t. Then we can describe $V(t)$ in terms of $A(t)$ and I_u using the following equation:

$$V(t+1) = \sum_{u \in A(t)} I_u(t - t_u) \qquad (Eq. 5.2a)$$

In the equation above, I_u denotes the influence function of the user u, which is a time series variable with a fixed length of L. The value of $I_u(t)$ denotes how much influence such the user u contributes to the diffusion process at timestep t. Here, limiting such user influence to a fixed window length of L means that a user can not influence other nodes after L timesteps since the time that the user performs some action to diffuse the information (such as posting a tweet, retweet, or like, etc.). Therefore, at the timestep t, user u interacts with a hashtag k, then he/she will influence others and trigger $I_u(t+1)$ diffusion rate at the next timestep $(t+1)$. Similarly, $I_u(t+2)$ will be trigger at the timestep $(t+2)$ and so on until the timestep $(t+L)$. However, after L timesteps, his/her influence will drop to 0. At this time, if that user continues to interact on the topic, a new cycle of influencing will be started.

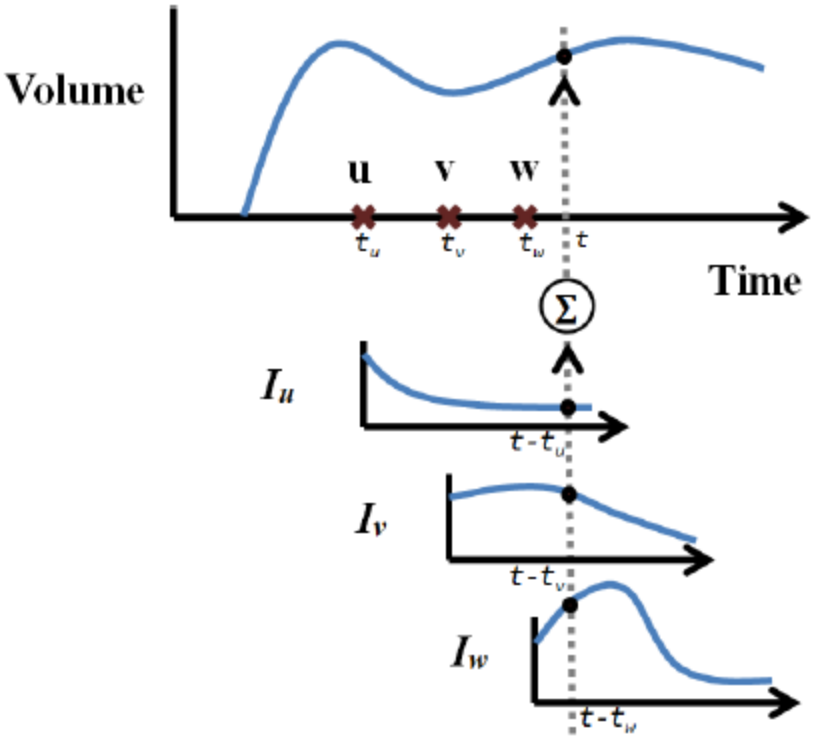


Figure 5.18 *The visualization of the relation of volume on individual user influence in the LIM model of Yang and Leskovec (2010)*

5.2.2 Methodology

This section describes or proposed user influence modeling and volume prediction system based on deep learning models (Nguyen, Hatua, Pothuraju, & Sung, 2018). Figure 5.19 displays the outline of such a system. The system starts with data collecting from Twitter and preprocessing. The objective of this step is to develop a user influence dataset. Next, the user influence model will be learned by the LIM model. On the other hand, our end-to-end deep learning model will capture the user influence functions as well as linking to the volume prediction layer at the end to forecast the global diffusion rate of different topics. The following subsections will describe in detail our proposed system.

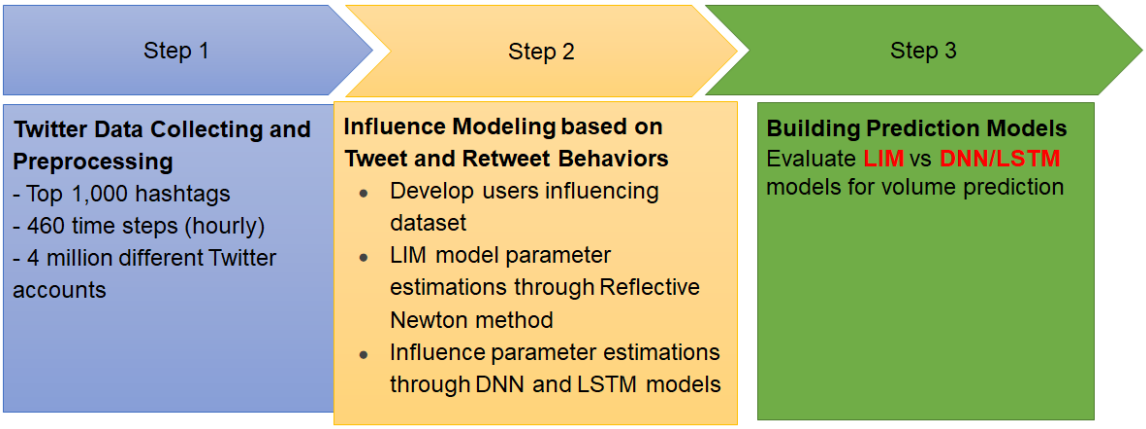


Figure 5.19 *The overview of our proposed framework to model the influence and build a volume prediction on Twitter*

5.2.2.1 Data Collection and Preprocessing

Data Collection. In the first phase of our proposed system, Twitter data of users' tweeting behaviors are collected. This step aims to build a time-series dataset of user influence on a list of interesting topics (hashtags). In this dataset, for each hashtag, at a time step, based on the tweet and retweet statistics of users, we can know how many newly infected user nodes (those who post the original tweets). Also, we know how many user nodes get affected by already active nodes (those who post retweets or replies on the previous posts of already existing nodes).

Data Preprocessing. This step will process the raw dataset that has been collected in the previous step to develop the training and validation dataset for the baseline LIM model and our proposed model. For the LIM model, two time-series datasets need to build: a) the Influence Indicator matrix, and b) the Volume matrix. Each cell in the Influence Indicator matrix represents the interaction of a user on a topic at a timestep (0 for doing nothing, and 1 for tweet/retweet/reply/like). The Volume matrix represents the total diffusion rate of all topics at the monitoring timesteps.

5.2.2.2 Baseline LIM Model

In this research, we will use the LIM (Yang & Leskovec, 2010) model as the baseline model for performance comparison. The LIM model is explained in detail below. Let consider a set of *N* user nodes and *K* different topics (hashtags). Any random subset of user nodes can interact with any topic over time. $V_k(t)$ is the volume of hashtag k at timestep t. In addition, let call $M_{u,k}(t)$ is the influence indicator of user u on a topic k at timestep t. Then $M_{u,k}(t) = 1$ if the user u interacts (tweets, retweets, replies) on the topic k at the timestep t, and $M_{u,k}(t) = 0$ otherwise. $I_u(t)$ is the user influence function of user u over time that need to be found. In the LIM model, $I_u(t) = 0$ if $t \geq L$, L is the maximum length of time that a user u can influence others from the time such a user performs some actions. Then, the volume $V_k(t+1)$ can be modeled as the sum of user influences I_u of those that interacted with such the topic k before time $(t+1)$:

$$V_k(t+1) = \sum_{u=1}^{N} \sum_{i=0}^{L-1} M_{u,k}(t-1) I_u(i+1) \qquad (Eq. 5.2b)$$

The above equation (Eq. 5.2b) can be represented in the matrix formula as $V = M.I$. Here, V is the global diffusion volume of all K topics in a total of T timesteps. Therefore, V has the shape of (K, T). Besides, M is the influence indicator matrix with the shape of (K, T, N, L). A cell (k, t, u, l) in the M matrix represents the interact of user u on topic k at the timestep $(t + l)$. Also, 'I' is the user influence matrix with the shape of (N, L). Figure 5.20 visualizes the shapes of different components in the LIM model. According to Yang and Leskovec, the equation Eq 5.2b can be solved to find the value of matrix *'I'* by using the Reflective Newton method after formulating that equation as a non-negative least square problem:

$$minimize \ \|V - M.I\|_2 \ with \ the \ condition \ I \geq 0 \qquad (Eq. 5.2c)$$

in which $\| \, . \, \|_2$ denotes the squared Euclidean norm operator.

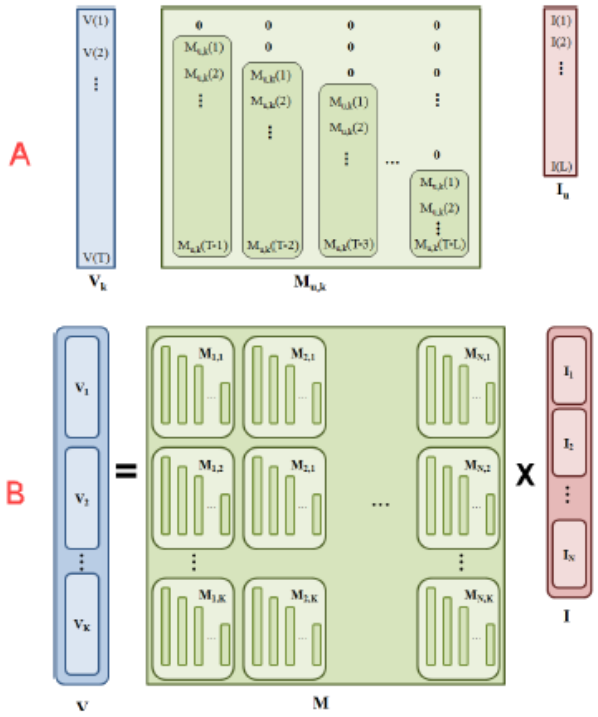


Figure 5.20 *The visualization of the LIM model for a single hashtag k (A), and all hashtags (B) (Yang and Leskovec, 2010)*

5.2.2.3 The Proposed Influence Modeling and Prediction Model

Motivated by the LIM model, we proposed end-to-end deep learning models to model and predict the global diffusion rate of topics. Compared to the LIM model, our models still have a similar formulation $V = M'.I'$, but the most significant changes are the new formulation of the user influence indicator matrix M' and the new user influence function I'. The global diffusion volume V is the prediction target for both of our models and the LIM model, so it will not be changed. Figure 5.21 displays the design of our proposed end-to-end models to model the user influence and predict the global diffusion volume.

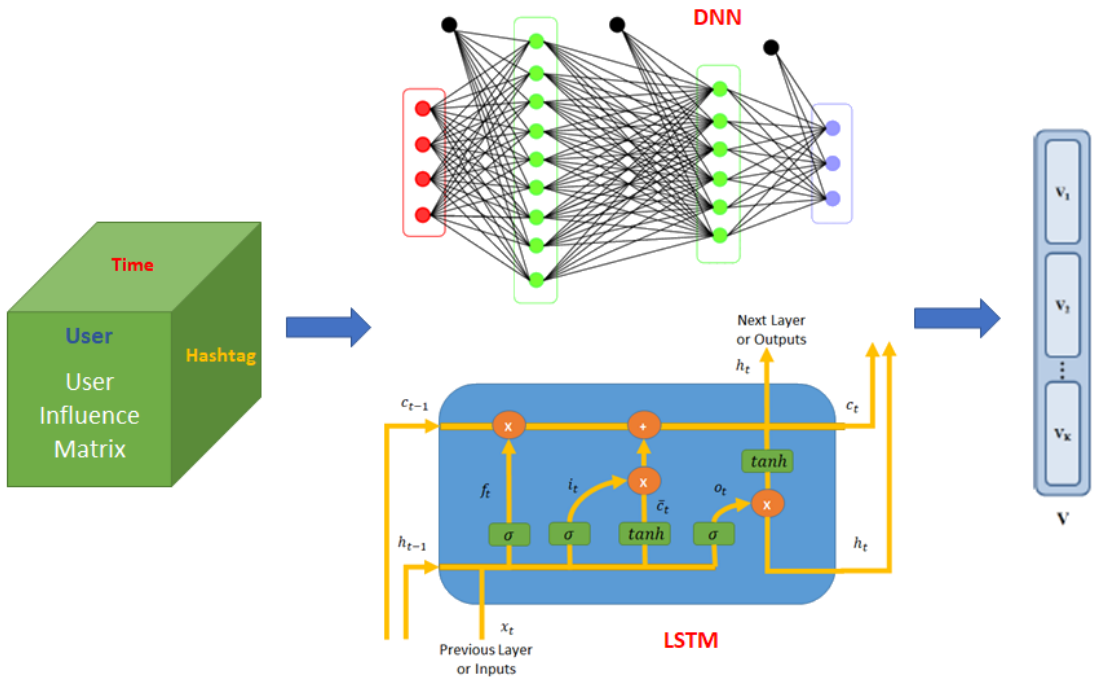


Figure 5.21 *Our proposed DNN based influence modeling and volume prediction model*

Compared to the LIM model, instead of modeling the user influence function as a fixed-length L of a time series variable, our models use a non-time limit user influence function $I'_u(k,t)$ that depends on topics and time. Our new user influence indicator matrix M' will drop the fourth dimension of fixed-length L of influencing time and replaced by a new dimension that describes the history of user influencing features. Therefore M' has the shape of (K,N,T,F), in which K is the length of diffusion topics, N is the size of user nodes, T is the length of monitoring timesteps, and F is the size of the newly introduced influence features. In our current design, the F dimension has a quantity of 3, which stores the normalized values of 3 elements. The first one is the ratio of newly created content (number of original tweets). The second one is the ratio of retweets (the total of retweeted posts of the user's posts on previous timesteps). Lastly, the last feature is the ratio of replies (the total of retweeted posts of the user's posts in prior timesteps). The value of each cell

of output dataset is the sum of all involve user's new tweets, retweets, and replies that corresponding to a specific time step and a hashtag. The M' matrix is visualized by the left green block in Figure 5.21.

The central components are the proposed deep learning models to study the user influence and predict the volume. Because our global diffusion volume prediction problem is modeled as a regression problem, therefore, in our model, the last Dense (fully connected) neural network layer use one hidden neuron and a linear activation. Between the input data M' and the last linear Dense layer, we employ three models to study the individual user influence:

- **DNN_24x266**: This model has two hidden layers, the first one has 24 hidden neurons, and the second one has 266 hidden neurons.

- **DNN_512x24x266**: This model has three hidden layers, in which 512, 24, and 266 are the corresponding number of hidden neurons for such three layers.

- **LSTM_128x64**: This model uses two stacked layers of LSTM cells, which have 126, and 64 cells correspondingly.

Scheme 5.1 displays the detailed layouts of the mentioned architectures above. The volume prediction at time step t is modeled as a linear regression of all users' influence parameter estimation values for relevant hashtags over time steps. Our proposed regression models are very similar to the volume estimation equation of the LIM model; however, the users' linear influence functions are replaced by a multi-layer (autocorrelated in case of LSTM) complex functions. Here, the weight parameters can be considered as the user influence parameters. In the case of using the LSTM model, besides the time series data of user influence, the volume values at previous time steps are also taken into account for

estimating the current output volume prediction value. That makes the prediction model even more complicated. The motivation here is LSTM is well known for effectively time series prediction.

5.2.3 Experimental Work

5.2.3.1 The User Influence Dataset

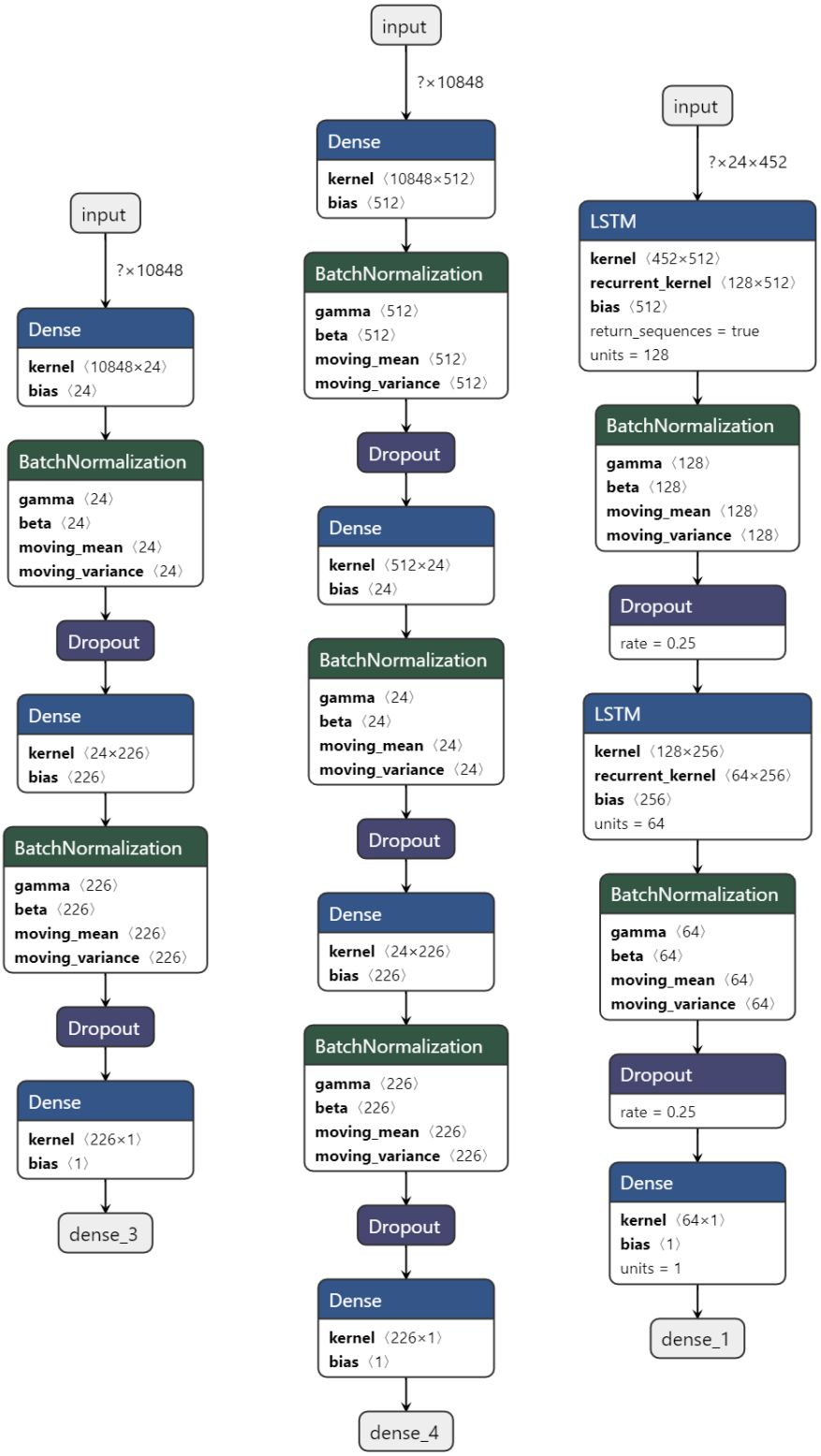
Following the method to collect data in the previous section (5.1.3), we also target Twitter for collecting raw data streams. We reused a list of 1,000 hashtags with the highest volume of tweets from our previous research. Later, we collected all tweets and users that related to such the above list in 3 weeks. Finally, our raw influence dataset contains about 14.5 million tweets and 4 million users who interacted with those 1,000 hashtags.

Next, the first data preprocessing step was executed on our raw dataset to remove unnecessary features, remove all hashtags and users with low tweeting activities (using the thresholds of 100 for hashtags and 20 for users). The following data preprocessing step is collecting the user influence time series and volume time series for each hashtag. We discretized our raw dataset in hour intervals, in which there are $T = 435$ hourly timesteps in our processed dataset. For each hashtag, at each timestep, we count the number of tweets, retweets, and replies of each user. Also, instead of using the window size $L = 10$, like in the original LIM paper, we extended the influence length to $L' = 24$. For the baseline LIM model, we constructed the M matrix following the described method in the original paper. The M matrix is a massive sparse matrix because, for a hashtag, there is a small subset of user nodes interacting with this hashtag.

With the assumption that hashtags are independent of each other, the set of all hashtags can be split into smaller groups of hashtags. Following this divide and conquer

approach, we reduce the size of the M matrix because, for each group, there is a smaller set of relevant users. In other terms, we break our original problems into multiple smaller problems, i.e., if we split the $K = 1{,}000$ hashtags into ten subsets $< K_1, K_2, \dots, K_{10} >$ of 100 disjointed hashtags. Then accordingly, we can have ten subsets of the corresponding volume $< V_1, V_2, \dots V_{10} >$ and ten subsets of the corresponding users $< U_1, U_2, \dots, U_{10} >$.

One of the essential parts of our proposed models is the construction of the new users' influence indicator matrix M'. Compared to the M matrix of the LIM model, this new matrix drops the influencing dimension L. We introduced the new dimension of the features F, in which two features need to be tracked: a) the number of newly tweets, and b) the number of retweets or replies. The first feature denotes how much the current user contributes to the diffusion of the topic and hence can be expected to influence other users in the future. Also, the second feature measures how much influence of the current users in the past. This latter feature is not the number of replies/retweets of the current user on a topic, but the number of replies/retweets of the current user's posts on the pasts. Here, the idea is the influence measurement of a user u at a timestep t can be estimated by how many retweets/replies of other users on the previous posts of the user u before t.



Scheme 5.1 *The three deep learning models for influence modeling and prediction*

5.2.3.2 Performance of the Baseline LIM Model

The baseline LIM model was implemented in MATLAB 2016. We used the Root Mean Square Error (RMSE) metric for comparing the prediction performance. The '**lsqlin'**[6] method in MATLAB was used to solve the non-negative least square equation $V = M.I$ of the baseline model to find the unknown user influence function I. One of the critical issues of this kind of implementation in MATLAB is handling the out of memory issues. This issue is caused by the big sparse matrix M which has the size of $K \times T \times N \times L$, with $K = 1000, T = 435, N \approx 4.0\ millions$, and $L = 24$. Therefore, we employed two following heuristics to solve that issue and improve the performance:

a) **Divide and conquer**: As mentioned in the previous section, with the assumptions that hashtags are independent, we split the set H of 1,000 hashtags into ten smaller subsets (H_1) to (H_{10}), each has 100 hashtags. For each subset H_i, we can collect the corresponding global diffusion volume V_i, the set of corresponding users U_i, and the influence indicator matrix M_i. U_i and M_i are much smaller compared to the original ones. The average number of users that interact on a group of 100 hashtags in our dataset is about 25,000.

b) **User grouping**: This approach is similar to the method that was introduced in the original LIM paper. To deal with the sparsity of the influence indicator matrix, for each subset of users U_i, we randomly grouped 100 users into a subgroup and represented a group by a consolidated user. In other terms, $U_i = \{G_i^1, G_i^2, \dots, G_i^m\}$, then we will a group of representing users $u_i^1, u_i^2, \dots, u_i^m$. Such the representing user will be considered for the collective behaviors of all corresponding users.

[6] **lsqlin**: Check the detail description here: https://www.mathworks.com/help/optim/ug/lsqlin.html

We performed 10-fold validation on our baseline LIM model, collected the results, and plotted its performance in Figure 5.22, 5.23, and 5.24. Figure 5.22 visualizes the average of user influence function during the monitoring window size of 24 hours. The visualization suggests a consistent pattern that the users' influence is most effective right after they interact on the topic (Hour 0 to 5) and gradually decrease over time. Moreover, their influence will increase from the last quarter of the 1-day window.

Figure 5.23 displays the performance comparison between the ground-truth values and the prediction values of the baseline model on the validation dataset over ten folds. Besides, such performance comparison for each fold is drawn independently in Figure 5.24. For easier performance comparison, we plot the negative values of our prediction results in these figures. Overall, the LIM model can predict well. However, it still misses at some points, especially where the global diffusion reaching the peak suddenly.

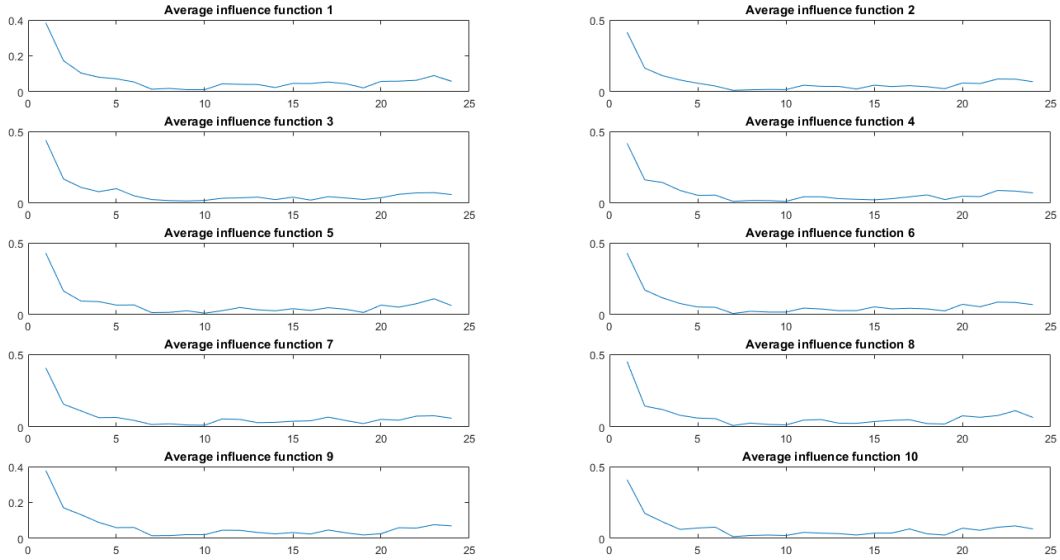


Figure 5.22 *The visualization of the average of all users' influence vectors for10-fold validations*

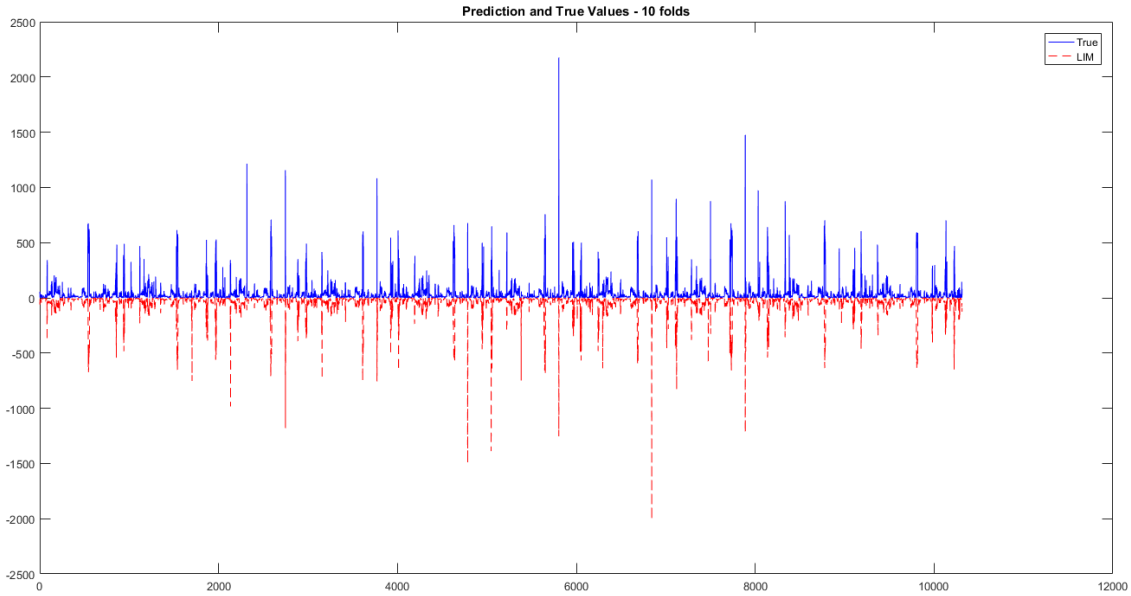


Figure 5.23 *The RMSE performance comparison of the LIM Model*

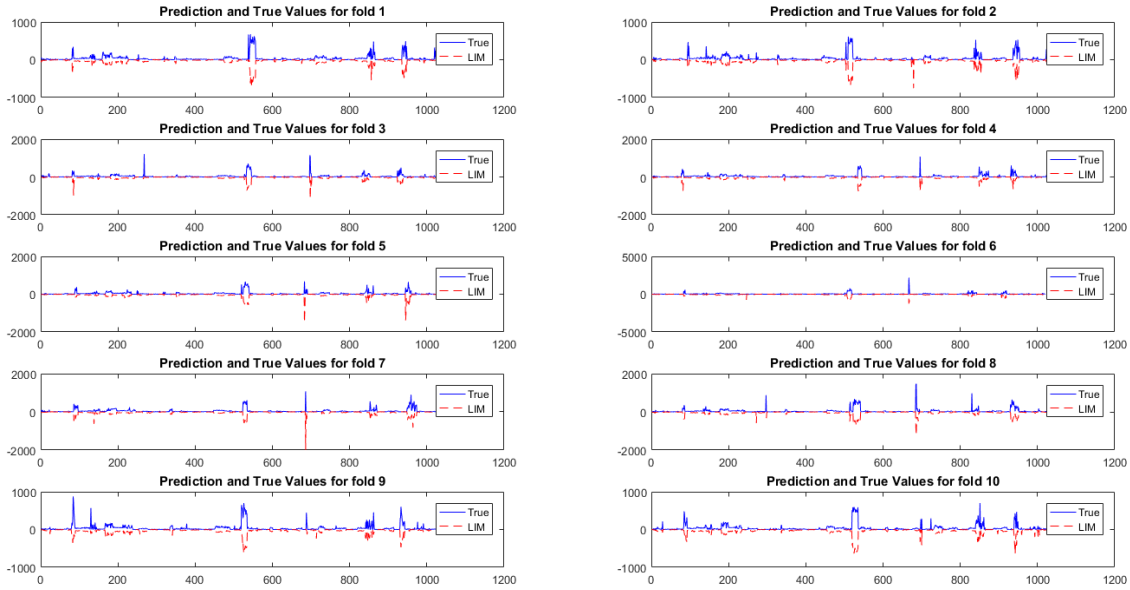


Figure 5.24 *The visualization of the prediction performance of LIM over 10-folds*

5.2.3.3 Performance of Our Proposed Deep Learning Models

Similar to the implementation approach of the LIM model, we also employed a hashtag splitting scheme and user grouping methods to reduce the complexity and the sparsity ratio of the user influence indicator matrix. In our experiment, we used three models (DNN_24x266, DNN_256x64x266, and LSTM_128x64) for modeling the user influence and building the global diffusion volume prediction. We implemented and performed our experiments using the Keras[7] platform. Regularization techniques, such as Dropout, Batch Normalization, Kernel regularizations (L1 and L2), were used in our models to avoid overfitting. Table 5.4 displays the best experiment settings for these three models in our experiments. Also, Figure 5.25, Figure 5.26, and Figure 5.27 present the RMSE performance comparison of our three proposed deep learning models.

Table 5.4 *The best hyperparameters of our deep learning models*

Parameters	**DNN_24x226**	**DNN_512x24x226**	**LSTM_128x64**
Architecture	Dense (24), Dense (226)	Dense (512), Dense (24), Dense (226)	LSTM (128), Dropout (0.2), LSTM (64), Dropout (0.2)
Loss function	MSE	MSE	MSE
Optimizer	Adam	Adam	RMSProp
Epochs	500	500	500
Batch size	256	256	256
Learning rate	0.001	0.001	0.001

[7] Keras: Check the Keras API at https://keras.io/api/

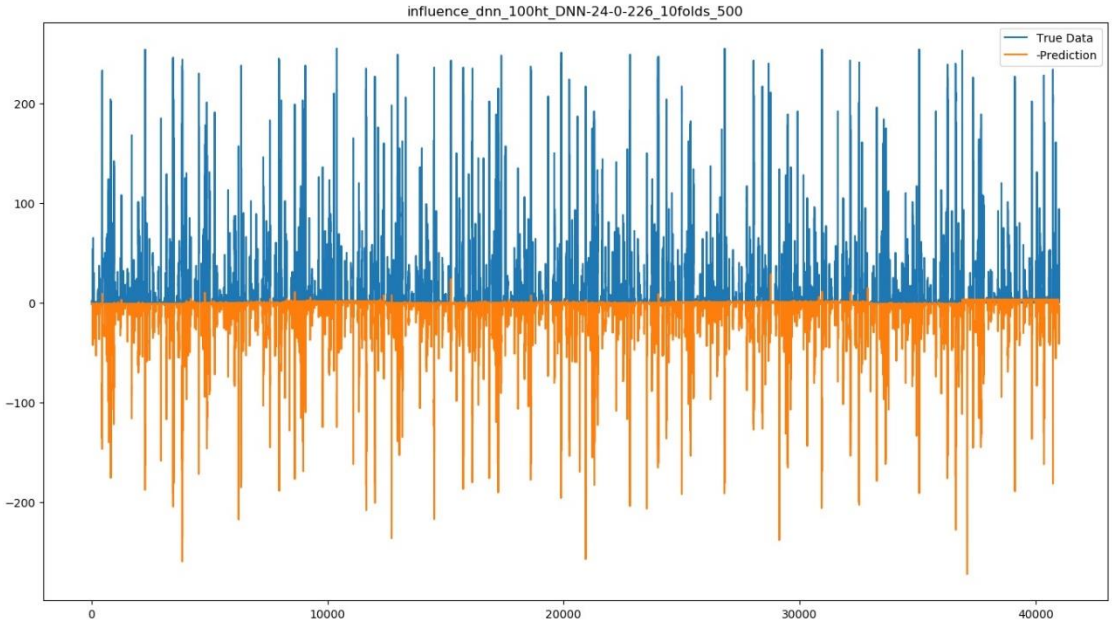


Figure 5.25 *The RMSE performance comparison of the DNN_24x226 architecture*

Table 5.5 presents the RMSE prediction performance comparison between the baseline LIM model and our three deep learning models. It is easy to see that all three proposed models outperform the baseline model. The LSTM model is the one who performs best with the RMSE of 12, which is about 0.63% error margin.

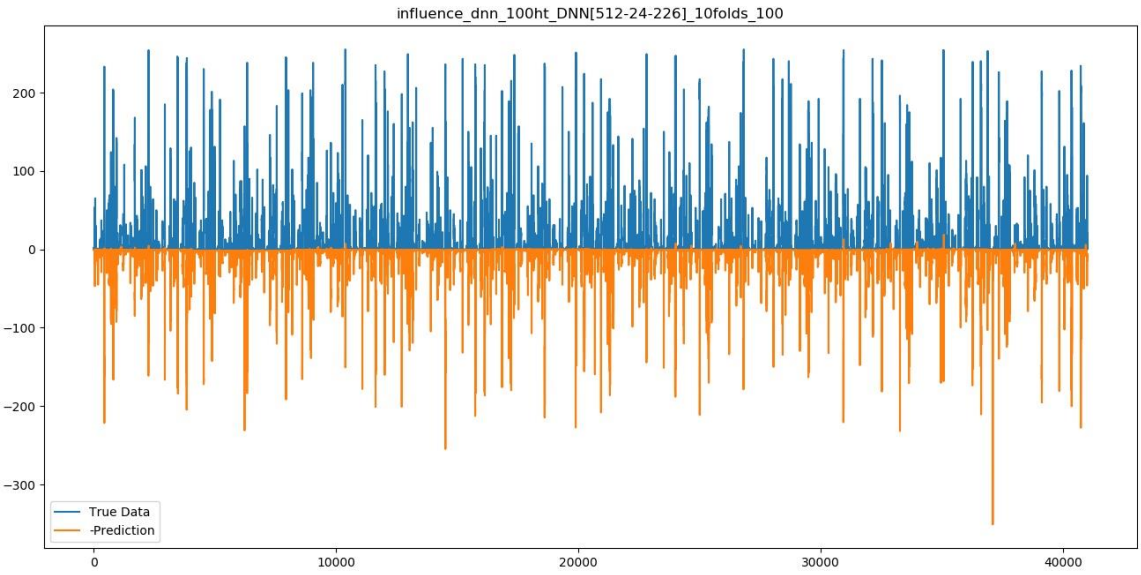


Figure 5.26 *The RMSE performance comparison of the DNN_512x24x226 architecture*

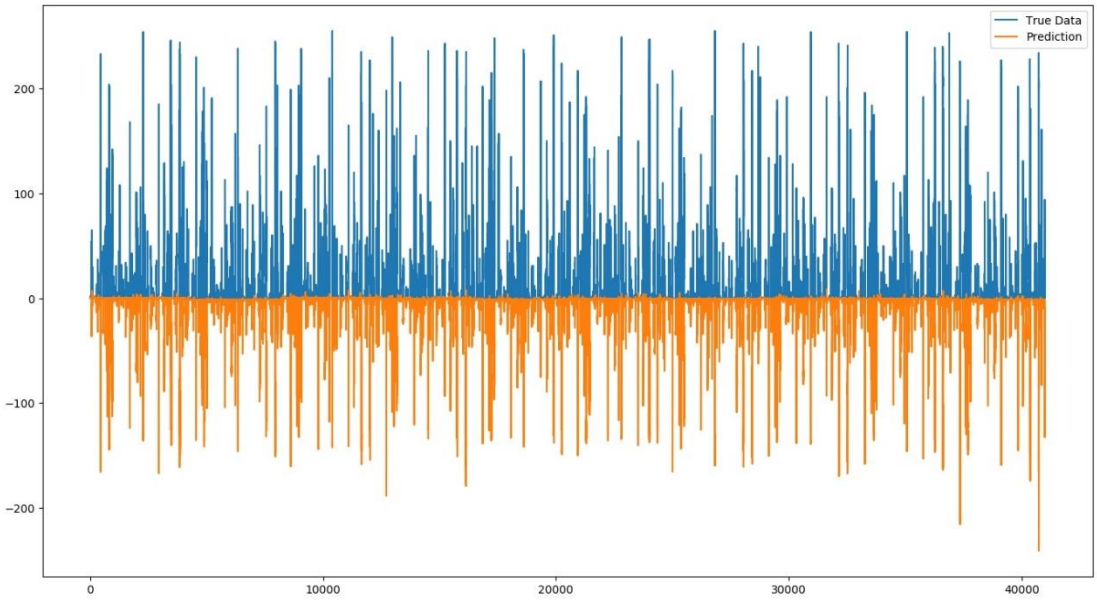


Figure 5.27 *The RMSE performance comparison of the LSTM_128x64 architecture*

Table 5.5 *The summary RMSE performance comparison between the baseline and our proposed models*

	LIM	**DNN_24x266**	**DNN_256x24x226**	**LSTM_128x64**
Fold 1	37.01	13.63	13.11	12.17
Fold 2	42.34	12.70	12.64	12.13
Fold 3	63.94	13.56	13.30	11.97
Fold 4	42.15	13.20	12.38	13.21
Fold 5	65.13	14.89	14.42	13.41
Fold 6	52.47	13.13	12.62	11.91
Fold 7	56.02	14.33	13.77	12.50
Fold 8	62.59	12.77	12.28	12.06
Fold 9	52.99	13.48	12.95	12.31
Fold 10	38.83	13.64	14.42	10.96
Mean	**51.35**	**13.53**	**13.19**	**12.26**
Error margin	**2.6%**	**0.70%**	**0.68%**	**0.63%**

This section introduced our end-to-end deep learning approach to model the influence and build the volume prediction for information diffusion on the social network.

The next section will present another study on user network interaction and topic diffusion patterns analysis on social networks.

5.3 Multi-Feed Weighted Topic Embeddings for Analyzing User Network Interaction and Topic Diffusion Patterns in Twitter

As discussed in section 3.2, the information diffusion processes depend on the content (popularity of topics, users' tweets, etc.), network structure, sentiment, and influence of related users, etc. My previous two studies have proposed a general time-series based approach and an influence-based approach for information diffusion modeling and prediction. In this study, the topic models and user network interactions are the targets of my research.

As discussed in section 3.5, multiple extensions of the LDA for the Twitter platform have been suggested to handle the problem of short-length tweets. Such extensions are the Twitterrank (Weng, Lim, Jiang, & He, 2010) algorithm or Author-Topic (Hong & Davison, 2010) model, the Twitter-LDA (Zhao et al., 2011) model, the Word Network Topic Model (WNTM) (Zuo, Zhao, & Xu, 2016). Most of these modeling approaches have focused on a single view of the topic distribution of related tweets rather than from the multiple viewpoints of user profiles.

User recommendation and retweet behavior analysis are two essential tasks in social networks. Section 3.5 has discussed different approaches to solve these tasks. Similar to topics modeling methods, the current methods of user recommendation and retweet prediction rely on the single view of the user's content or the TF-IDF features. In Natural Language Processing, embedding vectors have been used successfully in multiple applications, i.e., machine translation (Zou, Socher, Cer, & Manning, 2013) and sentiment

analysis (Tang et al., 2014). Motivated by these works, some approaches have been developed to generating the embeddings vector for multi-view data on Twitter. One such method is a unified graph representation of a multi-view of users' contents using the k-nearest neighbor algorithm (Greene & Cunningham, 2013). Another variant of the Generalized Canonical Correlated Analysis algorithm was introduced to generate the embeddings from the contents of Twitter users (Benton, Arora, & Dredze, 2016).

Contributions. A Multi-Feed Weighted Topic Embeddings (MFWTE) model is proposed to analyze the application of weighted multi-feed topic models for the tasks of analyzing user network interaction and topic diffusion patterns of Twitter users (Paudel, Hatua, Nguyen, & Sung, 2019). Our proposed method of separating the user feeds into multiple views based on their tweeting/retweeting behaviors, helped us capture the complicated temporal dynamics of Twitter and develop improved topic embeddings than a single view aggregated model. The following sections describe my study in detail.

5.3.1 Related Work

In 2010, a model used the Labeled-LDA (Ramage, Hall, et al. 2009) model to illustrate Twitter users using topic-models (Ramage, Dumais and Liebling 2010). They showed that the weighted mixture of the Labeled-LDA topic model and the TF-IDF distribution of user tweets improved the performance of the user-recommendation task. Also, Pennacchiotti and Gurumurthy examined the user-level topic modeling method for such a job by using an enhanced LDA model and replacing documents by streams of users' tweets (Pennacchiotti & Gurumurthy, 2011). Compared to the TF-IDF representations of users' tweets, their LDA system statistically had better performance and demonstrated the ability of user-level topic models for understanding user behaviors. However, in two works

above, they use a relatively small scale of evaluation size to evaluate experimental results (the numbers of positive/negative test users are 8 and 10, correspondingly).

5.3.2 Methodology

This section explains our proposed methodology to extract the multi-feed weighted topic embedding of user-topic tweets and profiles on Twitter (Paudel, Hatua, Nguyen, & Sung, 2019). Firstly, the method to collect and preprocess raw data is introduced, followed by the approach to extract our MFTWE models from different views, and finally, the method to evaluate our models on the tasks of user recommendation, retweet prediction, and topic diffusion patterns analysis. Our proposed methodology is presented in Figure 5.28.

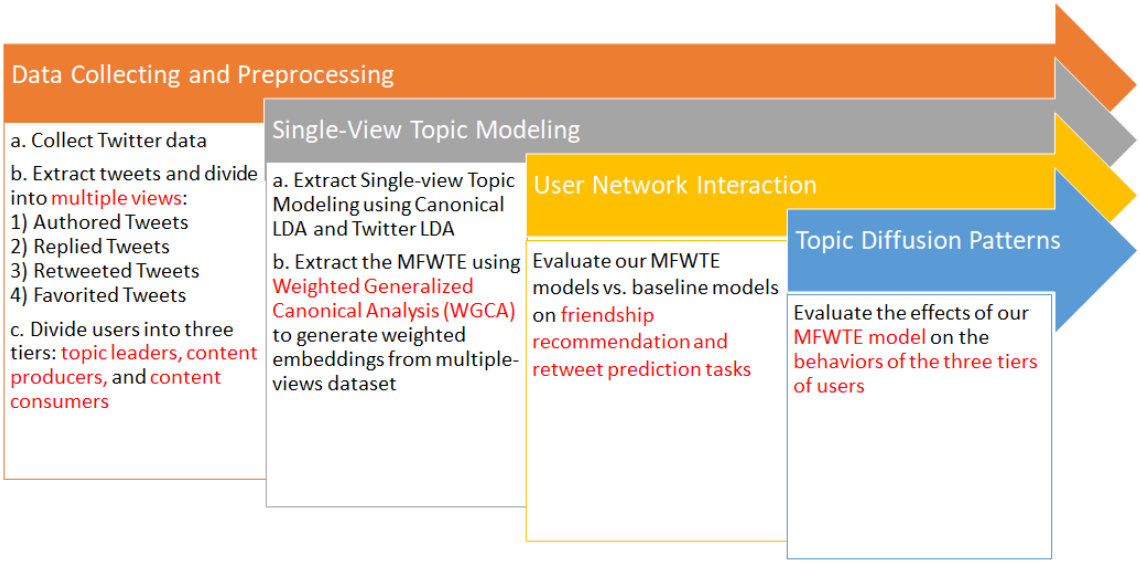


Figure 5.28 *The overview of our proposed methodology to extract MFTWE for studying user network interactions on Twitter*

5.3.2.1 Data Collection and Preprocessing

In Twitter, there is no explicit topic mechanism, but users can put hashtags in their tweets to mark the 'abstract' topics. Therefore, a topic in Twitter can be considered as a

group of related hashtags. In this work, we compiled a predefined list of five topics. For each topic, we collected sample data in one week and selected some most common hashtags related to such a topic. Table 5.6 presented the list of our exciting topics and their relevant hashtags. These are the seeds to collect the raw data streams of tweets and users' profiles.

Table 5.6 *The list of topics and related hashtags in this study*

Topic	**Hashtags**
Animal Rights	#animalrights, #animalabuse, #huntingkills, #saveanimals
Book Lovers	#amreading, #bibilophile, #booklover, #books, #bookworm
Domestic Violence	#domesticviolence, #meetoo, #sexualviolence, #violenceagainstwomen
Mental Health	#anxiety, #depression, #mentalhealth, #mentalIllness, #suicideawareness
Net Neutrality	#netneutrality, #savetheinternet

The first type of raw dataset to be downloaded is the set U_k of 2,000 users for each topic $k(1 \leq k \leq 5)$ in Table 5.6. To get this kind of dataset, we downloaded all tweets that contain the mentioned hashtags in Table 5.6 in one month (September 2018). Next, from such a raw dataset of tweets, we collected the list of associated Twitter users for each topic. We removed un-verified accounts to reduce the effect of bots. Also, we removed the non-English profile accounts. Next, we ranked them by their total tweets and selected ones that have at least ten tweets for the related topic. Finally, the most recent 400 tweets of each user were collected to create the authored tweets' view.

The second and third types of raw datasets are the sets of users' friends and followers. For each user u in the mentioned list U_k above, we collected a set V_u of their random 1,500 friends, and another set W_u of their random 1,500 followers. These raw datasets are used for friendship recommendation and retweet link prediction tasks. Finally,

for each user in the set of (V_u) or (W_u), we downloaded his/her most recent 400 tweets. Later, these raw datasets were used to develop the retweeted view, replied view, and favorite view.

Some data preprocessing techniques were applied to our raw tweets dataset to improve the quality of our topic models. Emoticons and special characters are familiar noise sources on Twitter and then were removed from our tweets. Also, we deleted non-English tweets, stop words, URLs, hashtags from the tweets. To reduce the effect of contraction words in the tweets, we searched and replaced them with full standard terms.

5.3.2.2 Extracting the MFTWE

In the scope of topic modeling methods on Twitter, we classified into three types of model: a) single-feed topic embeddings, b) multi-feed topic embeddings, and c) multi-feed weighted topic embeddings (MFTWE). Our proposed model falls into the last category (c). Next, we will describe the method to extract these embedding models from our raw dataset.

Single-feed topic embeddings: This is the baseline model for performance comparison with our model. To extract this kind of embeddings, we fed our processed users' tweets to Canonical LDA (the original LDA (Blei, Ng, & Jordan, 2003)) algorithm and Twitter-LDA (Zhao et al., 2011) algorithm. The WNTM (Zuo, Zhao, & Xu, 2016) model was not included in our experiment because of the high memory requirements to store the word co-occurrence matrix. The number of topics parameter was set to 6, while our datasets have five topics only because the Twitter-LDA requires one more background class. For both algorithms, there are two hyperparameters needed to be defined: α: the Dirichlet prior for Document-Topic Distribution parameter, and β: the Dirichlet prior for

Topic-Word Distribution. In our experiment, we used the same set of these parameters: $\{\alpha = 0.5; \beta = 0.01\}$. Moreover, we used Gibbs sampling to estimate the parameters of two single-feed embeddings baseline models.

Multi-feed topic embeddings: This is another kind of baseline model to be compared with our model. Compared to the previous single-feed models which take all user's feed as a single view, to extract this kind of embeddings, we need to split an individual user's tweets into four views that have been mentioned in section 5.3.2.1. The four views are: a) Authored view: the original tweets of the corresponding user, b) Retweeted view: the retweeted tweets of such a user on other users' tweets, c) Replied view: the replied tweets of such a user on other users' tweets, and d) Favorited view: the tweets of other users that such a user liked (favorited). In this work, we fed each view to the two mentioned topic modeling algorithms (Canonical LDA and Twitter-LDA) to get the single-view embeddings and finally combine them into a multiple-view embeddings model by concatenating. The same set of α and β hyperparameters were used.

Multi-feed weighted topic embeddings. Based on the idea that different view of users' feeds can contribute differently to the topic diffusion patterns of such users, we applied weighted parameters for the four view of users' tweets. We used the Weighted Generalized Canonical Analysis (WGCA) (Benton, Arora, & Dredze, 2016) algorithm to generate the weighted embeddings of our proposed model. WGCA is a variant of the Generalized Canonical Correlation Analysis (GCCA) (Carroll, 1968; Andrew, Arora, Bilmes, & Livescu, 2013; Arora & Livescu, 2014) algorithm. In our experiments, we employed the deep learning version of GCCA (Benton et al., 2019) and performed a grid search on the following hyperparameters: a) multiple view lengths = {15,20,40,100}, and

b) multiple view weights = {2,5,10,20,40,80}, to find the best combinations of the weighted parameters on our evaluation tasks.

5.3.2.3 Performance Evaluation

There are three kinds of evaluation tasks in our experiments: a) user friendship recommendation, b) retweet link prediction, and c) topics diffusion patterns of different hierarchical users.

User friendship recommendation: To evaluate our models on the friendship recommendation task, we used machine learning methods for constructing classification models of this task. Here, the input features are the topic embeddings of user profiles. The classification models will predict if two users (based on their topic embeddings) are friends or not to make a recommendation. Next, we need to develop training and validation datasets that contain the positive (friend) sample, and negative (non-friend) sample from the sets of users in our raw dataset (see section 5.3.2.1 for more detail). To do that, we randomly sample 100 target users from the list of users for each topic and put the friends of these 100 target users into the positive group. To get the negative (non-friend) samples, we randomly select the users who are followers or friends of the friends of these 100 target users, but not the friends of these target users. In other terms, if a user B is a friend of a user A in our list of 100 target users, B will be put into the set of positive (friend) samples. However, if a user C is a friend or a follower of the user B, but not a friend or follower of the user A, C will be put into the set of negative (non-friend) samples. We try to keep the number of positive and negative samples equal.

Retweet link prediction: Analogous to the user recommendation task, we used machine learning to build the classification models to predict the retweet link between two

Twitter users based on their topic embeddings profiles. The positive and negative samples of this task are defined as the following conditions. If a user B retweets more than ten tweets of a user A, then $B -> A$ is a positive retweet link between B and A. Besides, if B is a friend/follower of A, retweets less than ten tweets of A, but has direct retweet link with more than ten friends of A, then $B -> A$ is a negative retweet link. For this task, we select the negative samples as double to the positive samples.

Topic diffusion patterns analysis of hierarchical users: Based on the observation that each user has contributed differently to the topic diffusion processes, there are approaches to study the effects of different groups of users on such procedures. For example, in the Labeled-LDA user recommendation approach, the authors split the set of users into head, torso, and tail buckets based on the number of followers (Pennacchiotti & Gurumurthy, 2011). Different from them, we use the user-topic activities to split the users into three groups: a) Tier 1: the primary topic content creators, who spend more than 85% of their time interacting on the topic, b) Tier 2: the second-level topic content creators, who spend around 70-85% of their time interacting on the subject, and c) Tier 3: the third-level topic content creators, who spend about 50-60% of their time on the topic. The groups of users who fall below the third tier do not have a strong interest in a single subject but distribute their time on multiple topics equally. In our experiments, we observe the effects of our multi-feed weighted topic embeddings model on the behaviors of these three tiers of users.

5.3.3 Experimental Work

This section discusses the different results of our experiments. This section starts with the descriptions of our dataset, followed by the performance comparisons of our

MFWTE models compared to single-feed and multi-feed embeddings models on user friendship recommendation and retweet link prediction tasks. Ranked retrieval metrics, such as Precision @K, Recall @K, and the Mean Reciprocal Rank (MRR), are used to compare the performance between the three models. The last subsection will discuss the result of our study on our models on the topic diffusion patterns analysis.

5.3.3.1 Dataset Descriptions

To conduct a fine-grained study of our models on the user friendship recommendation, retweet link prediction, and topic diffusion patterns analysis, we split the training dataset and validation dataset into the three hierarchical tiers of users that are mentioned in the section 5.3.2.3. The data distribution of positive/negative samples of training and validation datasets for the user friendship recommendation and retweet link prediction tasks are displayed in Table 5.7 and Table 5.8 correspondingly.

Table 5.7 *Data distribution of training and validation datasets for user friendship recommendation task*

	Training dataset		**Validation dataset**	
	Friend (positive)	**Non-Friend (negative)**	**Friend (positive)**	**Non-Friend (negative)**
Tier 1	767	767	530	530
Tier 2	560	560	344	344
Tier 3	731	732	419	419

Table 5.8 *Data distribution of training and validation datasets for retweet link prediction task*

	Training dataset		**Validation dataset**	
	Friend (positive)	**Non-Friend (negative)**	**Friend (positive)**	**Non-Friend (negative)**
Tier 1	287	583	123	250
Tier 2	355	735	106	220
Tier 3	409	791	150	289

5.3.3.2 Performance Comparison of Friendship Recommendation and Retweet Prediction Tasks

This kind of performance comparison is conducted to demonstrate the advantage of using multi-view feeds over single-view feeds. The first experiment was the prediction performance of machine learning on the friendship recommendation and retweet prediction tasks. In this kind of experiment, we trained Support Vector Machine (SVM) using linear kernels on single-feed topic embeddings and our multi-feed weighted topic embeddings as input features. Also, we used the 10-fold Leave One Out cross-validation technique for evaluation. The detailed performance comparisons of these two tasks were presented in Table B.1 and B.2 in Appendix B. The performance comparison from these tables shows that the multi-feed embeddings outperform the single feed embeddings in the F1 score and accuracy metric.

The second experiment is the performance comparison between our multi-feed embeddings vs. the single-feed embeddings on ranked retrieval metrics. As mentioned before, the three metrics are Precision @K, Recall @K, and Mean Reciprocal Rank (MRR). For the friendship recommendation task, we compared the performance of our MFWTE model with two baseline models, which are the TF-IDF embeddings of user tweets and the single-feed topic embeddings. Figure 5.29 presents the Recall @K performance comparison between these topic embeddings models. It is easy to see that, in all three tiers of users, the multi-feed approach performs better when compared to the TF-IDF embeddings and the single-feed embeddings. Also, the evaluation of ranked retrieval metrics on the retweet prediction task was completed. For this task, we evaluated our multi-

feed topic embeddings with three baselines of the TF-IDF embeddings of user's tweets, topic embeddings of retweeted view of the user, and single-feed topic embeddings. Although the topic embeddings models outperform the TF-IDF embeddings, our models did not have a clear performance advantage over the single-feed and the retweeted view feed embeddings (Figure 5.29d). The detailed performance comparison is presented in Table B.3 in Appendix B.

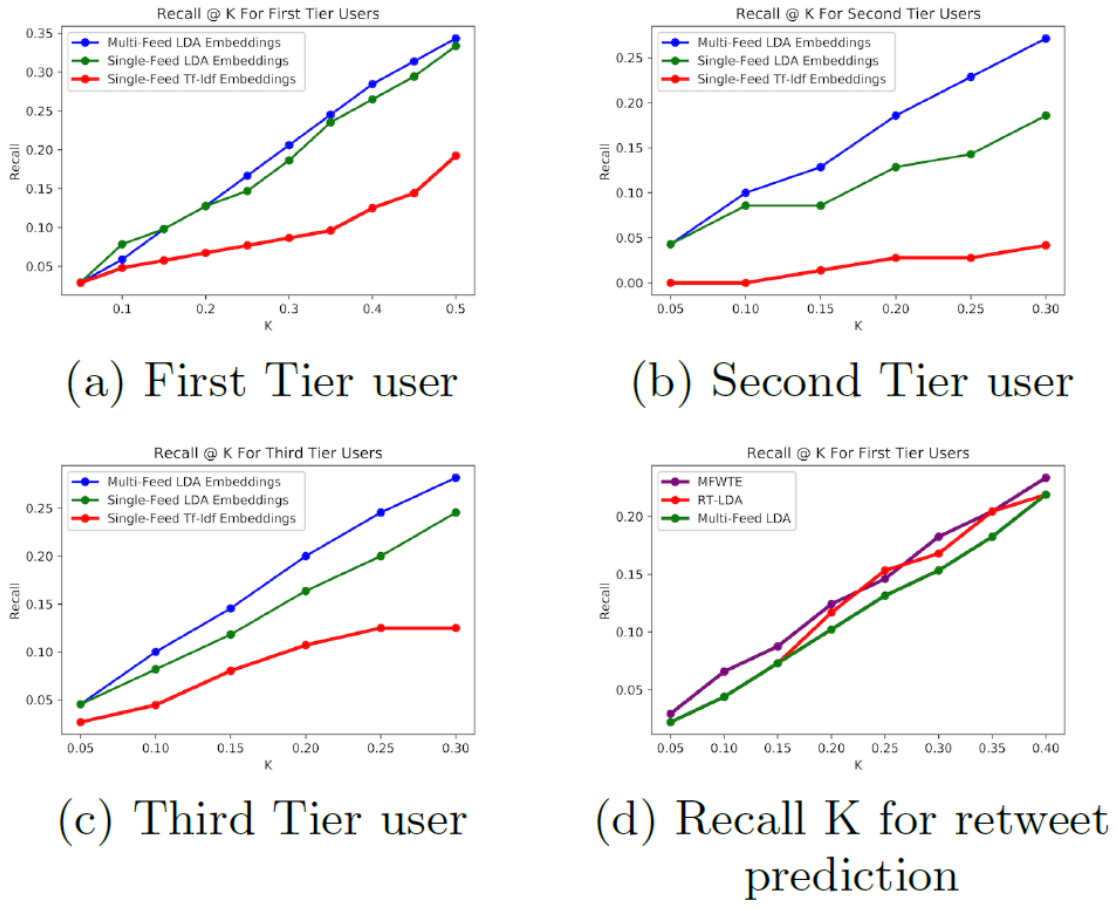


Figure 5.29 *The Recall @K performance comparison of friendship recommendation task on a) Tier 1 users, b) Tier 2 users, c) Tiers 3 users, and d) Recall @K of retweet prediction on Tier 1 users*

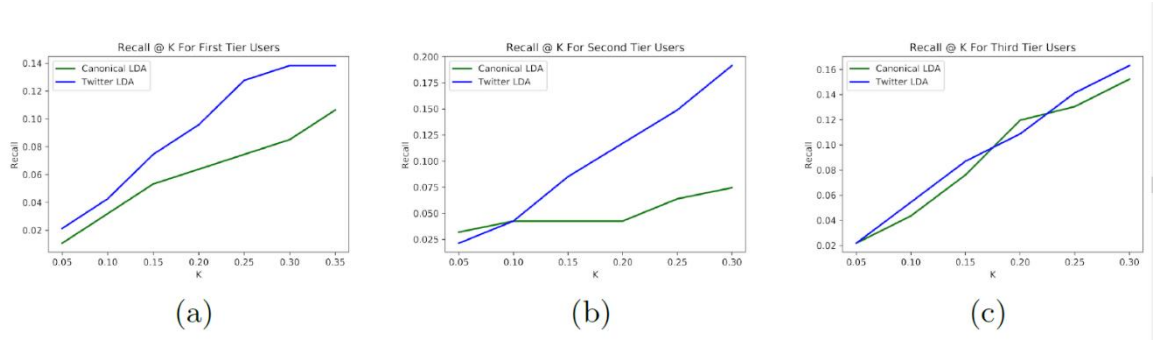


Figure 5.30 *The Recall @K performance comparison of Canonical LDA vs. Twitter LDA on Tier 1 users (a), Tier 2 users (b), and Tier 3 users (c)*

The third experiment is comparing the Recall @K performance of the two topics modeling algorithms: Canonical LDA and Twitter LDA on user-level tweets. The performance comparison displayed in Figure 5.30 suggests that the Twitter LDA is better than the Canonical LDA for all three tiers of users. This conclusion disagrees with the findings of Pennacchiotti and Gurumurthy (2011), who claimed that the tweet-level topic models are less efficient than the user-level topic models.

The fourth experiment is comparing the performance of our multi-feed weighted topic embeddings with other multi-feed non-weighted embeddings models. We believe that various tiers of users demonstrate a variety of behaviors during information diffusion processes, and the different views of users' tweets also have different features. To get the best optimal weights of our MFWTE models, for each tier of users, we performed the grid search of multiple view lengths = {15,20,40,100}, and multiple view weights = {2,5,10,20,40,80}. Then, we compared the rank retrieval metrics of our models with the multi-feed non-weighted topic embeddings model and the multi-feed TF-IDF topic embeddings model on the friendship recommendation and retweet prediction tasks.

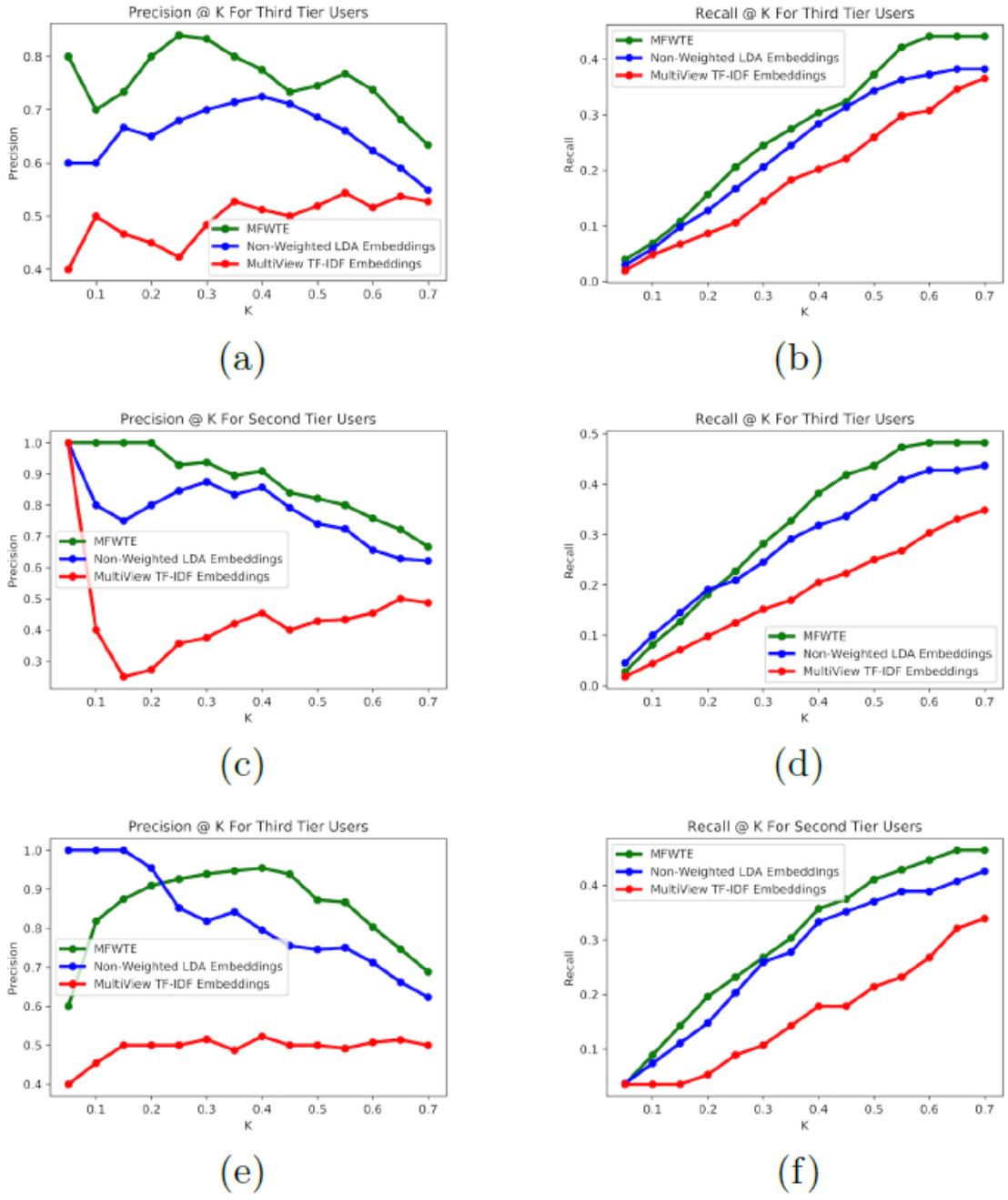


Figure 5.31 *The ranked retrieval performance comparison between our MFWTE models with the baseline models: a) & b): Tier 1 users, c) & d) Tier 2 users, e) & f) Tier 3 users*

The performance comparison on the friendship recommendation task in Figure 5.31 and Table B.5 shows that our MFWTE models outperform the other two baseline models. This result verifies our idea of the advantage of MFWTE over non-weighted

embeddings. Similarly, we also assessed the performance of our MFWTE models with the other two baseline models: the retweeted-view topic embeddings and the multi-view TF-IDF embeddings.

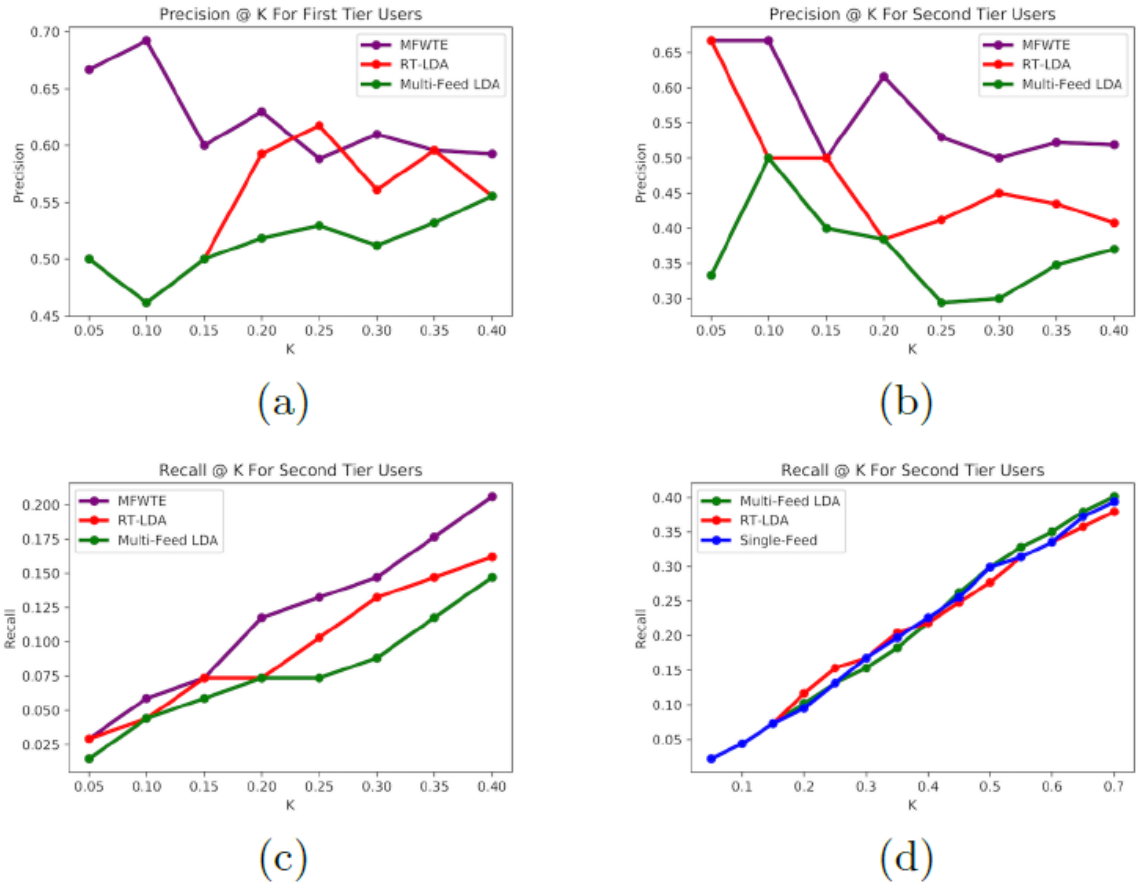


Figure 5.32 *The ranked retrieval performance comparison between our MFWTE models with the two baseline models on retweet prediction task: a) & b) Tier 1, c) & d) Tier 2*

The results in Figure 5.32 and Table B.6 show that our MFWTE also outperforms the baseline models on the retweet link prediction task in Tier 1 and Tier 2 users. However, in Tier 3 users, the retweeted view topic embeddings performed better than our models. Recall that the tier 3 users contribute 50-60% on the topic, which shows the unstable topics distribution compared to the other Tier 1 and Tier 2 users. That is the possible reason to explain this kind of result.

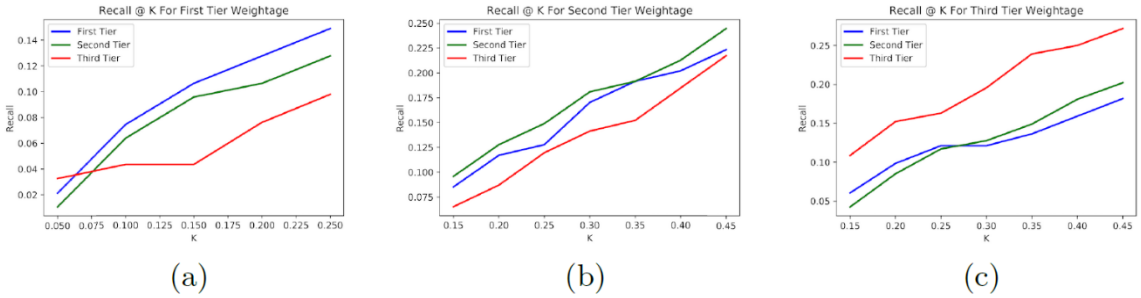
5.3.3.3 Performance on Topic Diffusion Pattern Recognition

The previous four experiments have proved the efficiency of our proposed models over previous approaches. This section discussed the results when we applied our models to analyze the topic diffusion patterns of three tiers of users on Twitter. In this fifth experiment, we used the three weighted combinations named C1, C2, and C3 for our studying of topic diffusion patterns. The configurations of those combinations are displayed in Table 5.9. The $\alpha, \beta, \gamma,$ and θ parameters are the weights for a) Author view, b) Replied view, c) Retweeted view, and d) Favorited view correspondingly.

Table 5.9 *The multi-view weighted parameters to study the topic diffusion patterns of three tiers of users*

Weighted combination	α	β	γ	θ
C1	1	40	1	1
C2	20	20	1	1
C3	1	1	20	20

Because the weighted combination C1 is biased to the replied view, so the idea is trying to study which tier of users engaging more time on responding to the tweets of the diffusion topics. Besides, the weighted combination C2 is biased to both authored view and replied view to the same extent. Lastly, the third weight combination C3 helps us to find our which tier of users engage more time on retweet and like other users' tweets. For each weighted combination in the set of C1, C2, and C3, we extracted the corresponding MFWTE models for each tier of users. Next, we tested these MFWTE models on friendship recommendation task to collect the ranked retrieval performance. Such performance is displayed in Figure 5.33 and Table 5.10.



(a) (b) (c)

Figure 5.33 *The ranked retrieval performance of our MFWTE models over three weighted combinations a) C1, b) C2, and c) C3*

The performance results in Figure 5.33a and the first column of Table 5.10 show that the users of tier 1 spent more time replying to the tweets than the other two tiers of users. Also, the Recall @K performance drops along with the decreasing level of user tier level. Regarding the performance of C2, the finding is the second tier of users are more likely to engage in creating tweets and replying tweets with the same rate. Another interesting finding on our topic diffusion patterns is the contribution of the third tier of users. According to the results, they are not engaging in the content creations of the topics but spent more time on the content distributions by retweeting or favoriting existing tweets.

Table 5.10 *The Recall @K performance of the three weighted combinations*

Tier of users	Recall @K of C1	Recall @K of C2	Recall @K of C3
Tier 1	0.14	0.22	0.21
Tier 2	0.12	0.24	0.22
Tier 3	0.09	0.21	0.29

5.4 Discussions

This section concludes the chapter on the introduction of different deep learning approaches to study the information diffusion processes on social networks.

In the first section of this chapter, I have introduced the methodology to model the information diffusion process using multivariate time-series (Hatua, Nguyen, & Sung,

2017). Our model captures the most attractive characteristics of diffusion analysis, which are volume, influence, and sentiment. Next, we can discover various clusters corresponding to multiple attributes in our model based on time series clustering algorithms. This pattern recognition has pointed out many useful temporal diffusion patterns in our dataset. Lastly, an efficient diffusion prediction model based on LSTM is introduced. The performance of such a model outperforms the traditional time-series forecasting method (ARIMA).

The previous study shows a promising data-centric framework to model, analyze, and predict the information diffusion process. However, the pattern recognition results on the influence and sentiments results can still be improved. In that study, user influence was studied based only on the number of direct/indirect followers, without considering the internal or external factor. Therefore, our second study focus on influence and sentiment analysis, which are two essential factors ruling the information diffusion processes (Nguyen, Hatua, Pothuraju, & Sung, 2018). In such work, a new deep learning methodology was developed for user influence modeling and predicting the global diffusion rate. Motivated by the LIM model, we used deep learning to build an end-to-end model that captures the individual impact of users, and quantitatively predict the global diffusion rate of information diffusion in the future. The performance comparison of our proposed model and the baseline LIM model shows some advantages of our approach, such as better performance, avoiding the out of memory issue (of the LIM model), modeling user influence using non-linear, and more complex functions.

In continuing the research in the information diffusion analysis domain, my third study proposed a Multi-Feed Weighted Topic Embeddings model. This model is deeply learned from the canonical correlation analysis on multiple views of users' tweeting

behaviors and profiles. The main idea is every user can be represented by an embeddings vector from historical data. The effection performance of our embeddings model on the friendship recommendation and retweet link prediction tasks has demonstrated the novelty of our approach. The last part of this study also shows different topic diffusion patterns that are shown by various tiers of users. Such patterns are, i.e., the first tier of users who are the topic leaders and contents creators, spend more time replying to the tweets besides posting new tweets. The second tiers of users spend almost equal time on both tweets and replies. The third tier of users contributes to the diffusion of the topic by retweeting rather than posting new tweets and replying.

CHAPTER VI – DEEP LEARNING FOR COLLECTIVE TASK LEARNING OF SWARM ROBOTICS

In this chapter, I will motivate and introduce different deep reinforcement learning approaches to handle multi-robot navigation and multi-robot foraging problems.

6.1 Multi-robot Navigation Problems

Section 4.2.1 has summarized the problem formulation and different approaches to the navigation problem of multi-robot. Because of the limited knowledge of the environment and other robots, many methods utilize both explicit and implicit communication channels to assist the robots with the optimal paths. However, for such a fundamental task like navigation, communication-less approaches are more preferred. There are two popular approaches to this problem. In the motion planning approach, there is a popular group of algorithms named the reciprocal velocity obstacles (Van den Berg et al., 2008; Van den Berg et al., 2010; Snape et al., 2011). Also, another group of algorithms is optimal reciprocal collision avoidance (ORCA) (Snape et al., 2010; Claes et al., 2012; Henness et al., 2012; Alonso-Mora et al., 2013; Bareiss et al., 2015; Godoy et al., 2016). These algorithms derive motion planning by finding the safe velocities that can avoid collisions. The drawbacks of these methods are the dependency on the perfect sensing assumptions and many parameters to be tuned. Section 4.2.1 also discussed different supervised learning and reinforcement learning approaches for this problem. These approaches are more robust compared to the motion planning approaches because they calculated the navigation actions directly from the robot's observations. However, some methods are designed only for single-robot, or the scenarios are not complex enough to be applied in the real world. Also, generalization issues have been discussed and addressed.

Contributions. A new cumulative approach, using transfer learning and adapted deep reinforcement learning algorithms, was proposed to develop the collision avoidance navigation system of multi-robot (Nguyen, Hatua, & Sung, 2019). During our cumulative training approach, the learning agents gradually improve their policy through the experiences from the least complex scenario to the most complex one. This approach is adapted with reward function shaping, transfer learning between agents, and multi-round learning paradigm. The core of the learning method consists of two modified policy gradient algorithm (PPO and TRPO). Our approach does not require direct communication between agents but only a communication channel with a master to receive the navigation tasks and the learning feedbacks during the training phase. Our proposed approach shows better performance compared to previous methods, reduces training time, and has better generalization performance in complicated indoor settings.

6.1.1 Methodology

Developing a robust navigation policy for swarm robotics is the main target of this study (Nguyen et al., 2019). The expected policy should generalize (or adapt with small effort) well to the different scenarios with short to long navigation tasks. Training an agent starting from scratch on a complex scenario is not easy and not work well. Curriculum learning (Bengio, Louradour, Collobert, & Weston, 2009) is a good strategy in this case, which is similar to the way humans and animals learn, starting from basic skills then complex skills. Motivated by the above idea, we develop our cumulative training methodology using transfer learning on the foundation of deep reinforcement learning to help the agents master the navigation skills from the easy to difficult ones. The subsequent subsections will explain the detail of such the methodology starting from the problem

formulation, the design of the low-level multi-agent reinforcement learning (MARL) layer, and the high-level cumulative training layer.

6.1.1.1 Multi-robot navigation problem descriptions

This study focuses on the collective navigation problem with the following conditions: a) the environment is bounded to prevent the risk of completely lost of robots, b) the robots cannot use their communication mechanism to exchange information with other agents for navigation tasks. Otherwise, they only communicate with a master controller to receive the tasks and the learning feedback during the training phase. This problem is formulated as a group of N robots moving concurrently and freely in a bounded environment. Each robot i starts from a source location s_i, navigate to a destination location g_i while avoiding the obstacles. There are two types of obstacles: the static ones (the fixed location objects such as a wall, table, chair, etc.) and moving ones (other robots). For simplicity, we only consider other robots as the moving obstacles in this problem. We use a low-profile robot for this problem, which is only equipped with proximity laser sensors to sense the distance with nearby obstacles. This low configuration will help our navigation approach more robust in real-world applications. At a timestep t, each robot i receive an observation vector o_i^t which consists of the 2D proximity laser sensor measurements, and the learning goal and feedback from the environment (the master). Every robot has the same policy that helps to compute the navigation action a_i^t for each robot i based on the observation o_i^t. The action a_i^t directs each robot from the current position p_i^t to the target place g_i during a fixed-length time (horizon) of Δt. Next, the steps of receiving new observations a_i^{t+1}, then generating new action a_i^{t+1}, are repeated until the robots reach their targets, or the total navigation time from the starting point is excess a threshold. The

objectives of the learning method are maximizing the success rate of all robots (the percentage of robots reaching the goals) and minimizing the mean arrival time of all robots while avoiding as much as possible the collisions.

Problem Formulation. The navigation problem of swarm robotics in this work can be formulated using the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) (Bernstein, Givan, Immerman, & Zilberstein, 2002). Following this model, the problem is defined using the tuple $< N, \mathbb{A}, P, \mathbb{O}, R >$. N stands for the number of agents. $\mathbb{A}$ is the agent properties: $\mathbb{A} =< S, O, A, \pi >$, in which S is the states set of agents, O is the set of observations, A is the set of possible actions, and π is the agent's navigation policy. P is the global state transition probability function: $P: S^N \times A^N \times S^N \rightarrow [0,1]$. $\mathbb{O}$ is the observation modeling function: $\mathbb{O}: S^N \times \{1,2,\dots,N\} \rightarrow O$, that maps the global observations to each agent's observation. In our approach, each agent receives its observation only without knowing the states or observations of other agents. Finally, R is the global reward function: $R: S^N \times A^N \rightarrow \mathbb{R}$, which defines the intermediate reward $R(s,a)$ if performing an action a on the state s. With the assumption of using the same type of robot, the robots can be assigned with the same navigation policy.

6.1.1.2 Design of the high-level cumulative training layer

The primary goal of this research work is to build a well-generalized collision avoidance navigation policy for swarm robotics. The policy should perform well in multiple scenarios in which consisting of short-range, medium-range, and long-range navigation tasks. Additionally, the policy should also be easy to adjust for unseen situations with very little training. Motivated by the idea of curriculum training and transfer learning, we proposed a cumulative multi-agent training framework (Nguyen et al., 2019). Such a

framework is presented in Figure 6.1, which contains two layers. The first and high-level cumulative training layer controls the learning process. The second and low-level is the multi-agent reinforcement learning (MARL) layer that takes the collected experiences from N robots agents and improves the policy. This section describes the high-level layer, while the next one explains the low-level layer.

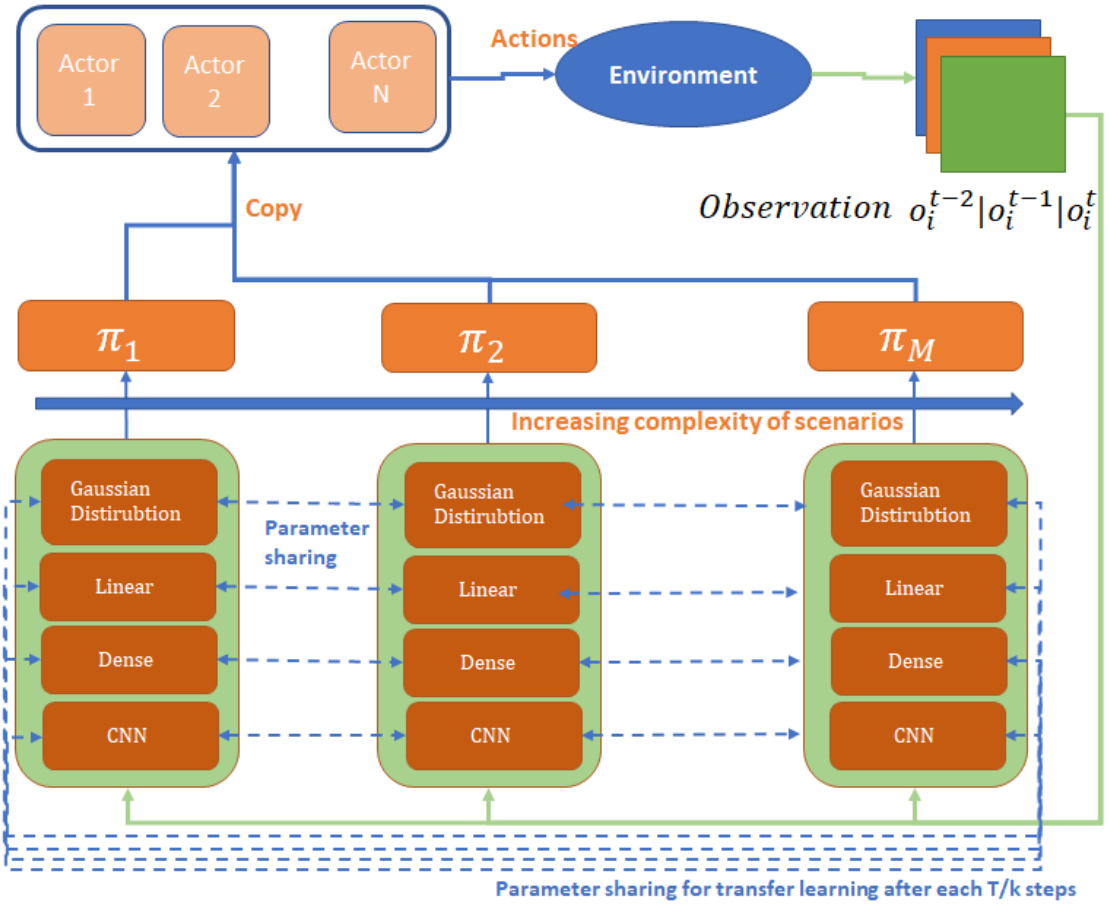


Figure 6.1 *The circular-like cumulative pipeline training paradigm*

There are M training scenarios in the environment, which range from the easiest one SC_1 to the most complex one SC_M. These scenarios will be introduced to the learning agents in order. This idea is mimicking the way humans and animals learning from basic skills to more advanced ones. The high-level cumulative training exploits the pipeline

approach with transfer learning through parameter sharing scheme to build and gradually improve the navigation policy from the increasing complexity of experiences. For each scenario, to train a policy, we use the number of required timesteps (T) and parameter synchronization time (k). The high-level controller starts two concurrent training processes. In the first step of the pipeline, the first process learns a policy $\pi_{1,1}$ on the scenario SC_1, while the second one learn a policy $\pi_{2,2}$ on the scenario SC_2. Both will learn in T timesteps and exchange the parameters of their models k times. Next in the second step, while the first process uses the existing policy $\pi_{1,1}$ to learns a cumulative policy $\pi_{1,2}$ for T timesteps on scenario SC_2, the second learns a new policy $\pi_{2,3}$ on the scenario SC_3 for T timesteps. They also exchange the parameters of their model k times during this second step. This kind of learning continues until the first process uses its previous policy $\pi_{1,M-1}$ to learn a cumulative policy $\pi_{1,M}$ for T timesteps on the last scenario SC_M, while the second process start to learn its policy $\pi_{2,1}$ for T timesteps on the first scenario SC_1. Now, we finish one round of cumulative training from SC_1 to SC_M.

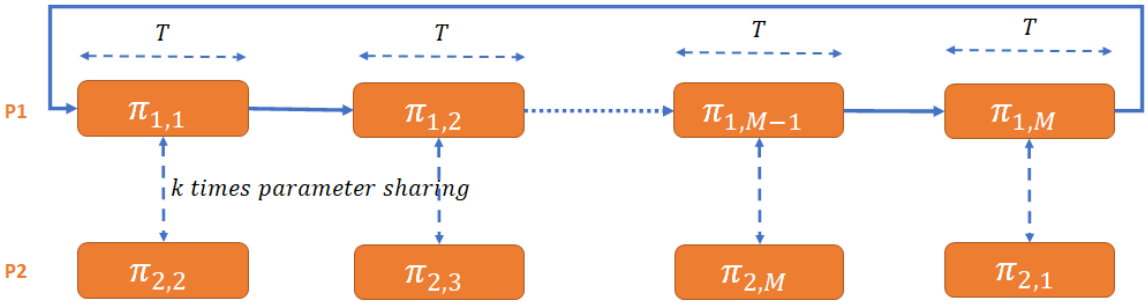


Figure 6.2 *The overview of high-level cumulative training flow*

The similar second round of training can be started from here with the first training process takes the policy $\pi_{1,M}$ to train again with scenario $SC1$ while the second process starts learning again $\pi_{2,2}$ on scenario SC_2. Figure 6.2 displays the idea of this high-level

pipeline training flow, where $P1$ and $P2$ denote two training processes, and $\pi_{1,i}$ and $\pi_{2,j}$ denote the corresponding internal policy in such processes. The final output cumulative policy of this high-level training layer will be the policy in the first process.

The next section will explain the detail of the low-level MARL layer.

6.1.1.3 The low-level MARL layer

This layer follows the typical setup of a multi-agent reinforcement learning paradigm. For this navigation problem, we make use of the "*centralized learning and decentralized execution*" scheme. The idea of this scheme is to give the learning agents the ability to access additional information during training but not in execution time. Therefore, this idea can help the agents to learn better by overcoming the non-stationary environment problem. In our approach, we limited the additional information that an agent can receive to make our solution easy to adapt in the execution phase. In this low-level MARL layer (Nguyen et al., 2019), there are N actors, which are the N robot agents. Each actor will receive the same navigation policy from the training layer. Based on the current observations, these actors generate actions by executing their policies and interact with the environment in a decentralized way. Next, these actors receive new observations and feedback from the environment to form new experiences, including the previous observation states, past actions, current observation states, rewards, and other additional information. These experiences will be collected from all actors and feed again to the centralized learner to update the navigation policy. The following subsections will describe in detail our multi-agent reinforcement learner approach.

Observation Model. The observations that an agent receives after acting are consisting of the 2D laser sensor measurements and the learning feedback from the

centralized master during training phrase. The robot prototype of our experiments is equipped with a ring of 6 laser sensors on top that together cover the entire surrounding area of 360 degrees. These proximity sensors can sense maximum obstacle distance up to 5 meters. Therefore, the first type of observation in our model (o_z), is the sensor measurement, which is a vector of length of 360. The second type of observation o_v in our model is the actual current translational and rotational speed in m/s. This observation is used because our policy will sometimes create out-of-bound actions which are adjusted by the system later. The third type of observation (o_g) is the relative distance and angle to the goal position projecting on a robot's coordination system. At a timestep t, a robot i will receive an observation vector $o_i^t = [o_{i_z}^t, o_{i_v}^t, o_{i_g}^t]$. To help our navigation policy (explained later) in detecting the actions and states of neighbors, we stack the last three observation vectors instead of a single timestep vector. Therefore, the final observation vector that an agent i receives is $[o_i^{t-2}, o_i^{t-1}, o_i^t]$. Furthermore, the laser measurement vector is normalized into [0,1] by dividing with the maximum sensing distance of obstacles. It is noted that our observation model is continuous.

Action Model. The robot prototype in our system uses the differentiate driven kinetics mode. At the timestep, the robot receives an action consisting of two components: the translational speed and rotational speed. In our experiments, the range of the translational speed is set to $[-1.0,1.0]$ in the scale of m/s while the range of the rotational speed is set to $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ in the scale of $radians/s$. The negative range of translation speed is aiming for the case the robots need to go backward for collision avoidance or recovering from a stuck situation. Our action space is also continuous.

Reward Model. The reward function design is crucial in any reinforcement learning system because of its role in guiding the learner to optimize the policy. At a timestep t, the reward signal $r_i(t)$ of a robot i is defined by the following equation:

$$r_i(t) = r_i^g(t) + r_i^m(t) + r_i^p(t) + r_i^c(t) + r_i^v(t) \qquad (Eq. 6.1a)$$

Our reward model consists of five different components. The $r_i^g(t)$ is the reward the robot i receives when reaching its final goal position. The $r_i^m(t)$ is the reward when the robot i reaches some milestones in their way to the final goal position. That is the way we design the navigation task to the robots. If we provide only the final goal position to the robots and let them explores and find their goals. In such situations, especially in the case of complex and long-navigation tasks, the robots have the problem of achieving their goals. Therefore, we generate checkpoints (virtual landmarks) on some possible paths of each robot from the start location to the target place. Next, the $r_i^p(t)$ is the reward/penalty regarding the progressing to the target. Also, the $r_i^c(t)$ is the penalty for making any type of collision. Finally, the $r_i^v(t)$ is the penalty for making an illegal move (out-of-bounds translational and rotational speed).

It is straightforward to calculate the $r_i^g(t)$. We use the Euclidean distance $d_i^g(t)$ from the present position to the goal position of the i^{th} robot for this calculation. If such a distance is small enough (less than a defined short distance δ_d^g), we assign a fixed reward $r_{arrival}$ for the goal-reaching state: $r_i^g(t) = r_{arrival} \;\; if \; d_i^g(t) < \delta_d^g$. Similarly, the intermediate reward $r_i^m(t)$ is based on the Euclidean distance of the present location with the nearest milestone in the set of possible paths.

The progress reward $r_i^p(t)$ is the small reward or penalty $r_{progress}$ based on the current relative location of the robot to its target position. In this model, the robot will receive a positive reward if they reduce the Euclidean distance with its goal compared to such the distance in the previous timestep and vice versa. The equation to calculate progress reward is: $r_i^p(t) = r_{progress} \times \left(d_i^g(t) - d_i^g(t-1)\right)$.

There are two types of collision can happen, which are the collision between robot agents and the collision between the robots and the static obstacles. To examine the effect of these two types of collisions, we define two different safe distances which are the safe distance between two agents δ_d^r and the safe distance between an agent and obstacles δ_d^o. Later, we extend this concept to add a small penalty if the distance of a robot to nearby objects is close to these safe distances. There is a fixed penalty number for the existence of any collision r_{col}. The equation to calculate the collision penalty $r_i^c(t)$ is:

$$r_i^c(t) = \begin{cases} r_{col} & if\ d_i^r < \delta_d^r\ or\ d_i^o < \delta_d^o \\ 0.1r_{col} & if\ d_i^r < 2\delta_d^r\ or\ d_i^o < 2\delta_d^o \end{cases}$$

The last type of penalty is dealing with action values. Although negative speed is allowed in our action model, to encourage the robot to go forward, we apply some small penalty if the robot moves backward. Also, to stimulate the agility driving of the robot (not making turn suddenly), we apply a minor penalty for making such large rotational speed by using a predefined rotation threshold δ_v^a. The equation to calculate the speed penalty is:

$$r_i^v(t) = r_{linear} \times |v_i^x(t)| + r_{angular} \times |v_i^a(t)| \quad if\ v_i^x(t) < 0.0\ or\ v_i^a(t) > \delta_v^a$$

in which $v_i^x(t), v_i^a(t)$ are the translational speed and rotational speed of robot i at timestep t. $r_{linear}, r_{angular}$ are the small penalty for out-of-bounds translational speed and rotational speed correspondingly.

Algorithm 1: Multi-Agent TRPO (modified from Schulman et al., 2015)

Initialize policy π_0, the value function V_0 and hyperparameters
for k=0, 1, 2, … do
  for t=0, 1, 2, …, T_{mel} do (T_{mel}: maximum episode length)
    run policy $\pi_k = \pi(\theta_k)$ to sample action vector a_k^t to collect the set of trajectories $D_k = \{\tau_i\}$ of all robot agents ($1 \leq i \leq N$)
    Compute the extended rewards-to-go $\hat{R}_N^t$ for N robots by aggregation
    Computer the advantage estimates $\hat{A}^t$ based on V_k
    Estimate policy gradient $\hat{g}_k$ (see [20] for more details)
    Apply a conjugate gradient to compute $\hat{x}_k$
    Update the policy π_{k+1} by backtracking line search
    Fit value function by regression on mean-squared error (MSE) via gradient descent
end for

Scheme 6.1 *The pseudocode of the adapted-TRPO algorithm (Nguyen et al., 2019)*

Algorithm 2: Multi-Agent PPO (modified from Schulman et al., 2017)

Initialize policy π_0, the value function V_0 and hyperparameters
for k=0, 1, 2, … do
  for t=0, 1, 2, …, T_{mel} do (T_{mel}: maximum episode length)
    run policy $\pi_k = \pi(\theta_k)$ to sample action vector a_k^t to collect the set of trajectories $D_k = \{\tau_i\}$ of all robot agents ($1 \leq i \leq N$)
    Compute the extended rewards-to-go $\hat{R}_N^t$ for N robots by aggregation
    Computer the advantage estimates $\hat{A}^t$ based on V_k
    Update the policy π_{k+1} by maximizing the PPO-Clip objective via stochastic gradient ascent with Adam optimizer
    Fit value function by regression on MSE via gradient descent
end for

Scheme 6.2 *The pseudocode of the adapted-PPO algorithm (Nguyen et al., 2019)*

Navigation Policy Training Methods. In this research work, two adapted for multi-agent settings policy gradients algorithm are used to train our multi-robot navigation policy. The two policy gradients algorithms are based on the PPO and TRPO algorithms (see section 2.6 for more detail). As described in the previous subsection, the high-level

cumulative training processes will either choose adapted-PPO or adapted-TRPO as the low-level MARL learner. The learner process will be executed in the "*centralized learning, decentralized execution*" fashion. It means the current navigation policy is broadcast to all robot agents (actors) to interact with the environment where one of a training scenario is deployed. The agents will collect all new experiences, feed to the central learner to update the navigation policy. This process will be repeated in many training episodes, in which each episode is defined by a threshold of maximum episode length (T_{mel}). The robots are given a maximum T_{mel} timestep to reach their goal. After an episode finish, the environment will reset the robots to their initial positions. The initial position of the robots is selected randomly at the beginning of each cumulative training step and different for each scenario. The two adapted PPO and TRPO algorithms are described in the following diagrams of Scheme 6.1 and Scheme 6.2.

Navigation Policy Neural Networks Models. Our neural-networks navigation model is built using the combination of CNN layers and Dense layers. We make use of two Convolutional 1D (Conv1D) layers to do the representation learning of the normalized of 2D laser sensor measurements $[o_{i_z}^{t-2}, o_{i_z}^{t-1}, o_{i_z}^{t}]$. Next, the latent output vector resulted from the two above layers will be flattened and concatenate with the two other observation vectors $[o_{i_v}^{t-2}, o_{i_v}^{t-1}, o_{i_v}^{t}]$ and $[o_{i_g}^{t-2}, o_{i_g}^{t-1}, o_{i_g}^{t}]$ to form a merged latent input vector to the next three Dense layers. The last Dense layer has two neurons and used 'linear' activations. Then, the outputs from this layer are used to create a 2×2 multivariate Gaussian distribution $(v_{mean}^{t}, v_{logstd}^{t})$. This distribution will be used to sample the actions a^t for N robots. The sampled actions will finally be clipped into the valid range of the translational

velocity and rotational velocity. Table 6.1 displays the detail of our proposed multi-robot neural network navigation policy.

Table 6.1 *The stacked neural networks architecture of the navigation policy*

Layer	Input	Descriptions
Conv1D_1	$[o_{i_z}^{t-2}, o_{i_z}^{t-1}, o_{i_z}^{t}]$	Conv1D: 64 kernels, filter size=5x5, strides=2, ReLU activation
Conv1D_2	Conv1D_1	Conv1D: 64 kernels, filter size=3x3, strides=2, ReLU activation
merged	Flattern(Conv1D_2) + $[o_{i_v}^{t-2}, o_{i_v}^{t-1}, o_{i_v}^{t}]$ + $[o_{i_g}^{t-2}, o_{i_g}^{t-1}, o_{i_g}^{t}]$	Concatenation layer
Dense_1	merged	Dense (outputs=256, activation=ReLU)
Dense_2	Dense_1	Dense (outputs=128, activation=ReLU)
Dense_3	Dense_2	Dense (outputs=64, activation=ReLU)
Dense_4	Dense_3	Dense (outputs=2, activation=linear)
m_Gaus	Dense_4	mean = Dense (outputs=2, activation=linear) logstd = Dense (outputs=2, activation=linear)

6.1.2 Experimental Work

This section will describe our experiments and discuss different results from our experiments. This section starts with the experiment setup, which the simulation environment, hyperparameters settings, experiments, and performance metrics are described, followed by the performance comparison of our approach with the baseline methods.

6.1.2.1 Experimental Setup and Hyperparameter Settings

Simulation Environment. In this research work, we implemented our experiments using the Player/Stage (Gerkey, Vaughan, & Howard, 2003) robot simulator[8]. Six different scenarios have been used in our experiments. These scenarios are mimicking real-world

[8] The related documents and source codes of Player/Stage can be accessed here: http://playerstage.sourceforge.net/

indoor environmental settings such as apartments, office buildings, factories, or warehouses. The six scenes from $SC1$ to $SC6$ are ordered by the complexity of the scenarios as required by our cumulative training approach. Figure 6.3 displays the maps setting of those scenarios. While simple, short to medium navigation tasks are designed in the first three scenarios (SC1, SC2, SC3), the medium to long-range and more sophisticated navigation ones are implemented in the last three scenarios.

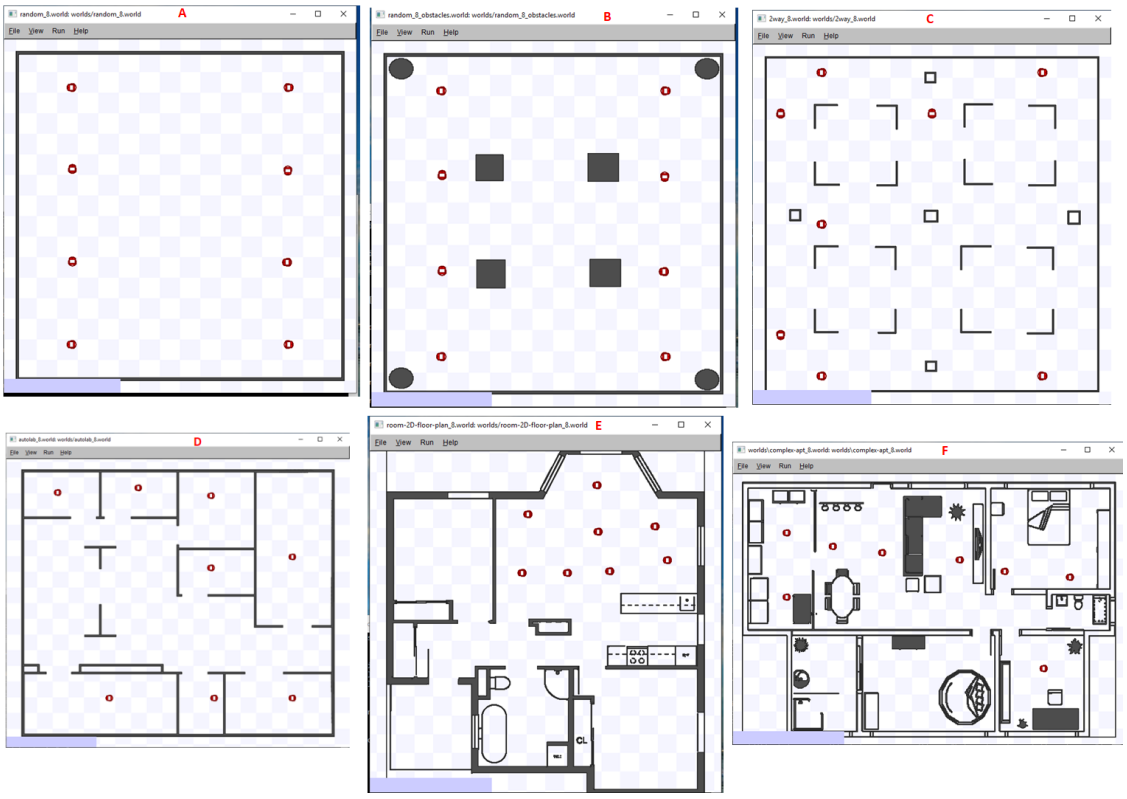


Figure 6.3 *Six scenarios in our experiments: A) SC1, B) SC2, C) SC3, D) SC4, E) SC5, and F) SC6*

Our robot prototype in our experiments is based on the Pioneer P3-DX model. That is a lightweight two-wheel two-motors robot with differential-drive kinetics. While the maximum translational speed of this robot is $1.2m/s$, and its maximum rotational speed is $300°/s$. For agility, in our experiments, the translational speed is limited to $[-1,1]$ m/s,

and the rotational speed is limited to $[-\frac{\pi}{2}; \frac{\pi}{2}]$ radians/s. Similar to the scenarios, this model of the robot is also built on the Player/Stage simulator. The simulator provides a subscriber-mode Python binding library to communicate with the robot. Our Python MARL learner use this Python robot binding library to send actions and receive the observations.

Experimental Setup. Two circular-like cumulative training experiments, each with adapted-PPO and adapted-TRPO, were performed. During the training processes, for each scenario maps, we generated different random initial and goal locations for each robot for a better sampling of the learning experiences. For performance comparison, we also performed two baseline experiments that trained using either PPO or TRPO algorithms in each scenario. The baseline policies are expected to work well on the situation that they are trained but not on the others. Our cumulative trained policies will be evaluated in each scenario with the corresponding baseline policy.

Performance Metrics. Both the cumulative policy and the baseline policy will be evaluated in each scenario for ten episodes. Two performance metrics, which are episode success rate and mean episode arrival time, are used in our experiments. The episode success rate measures the proportion of robots that can reach their goal. The mean episode arrival time measures the average of timesteps of robots to reach their goal, without considering the ones who are still on their ways.

Hyperparameter Settings. In our experiments, the horizon time, the time for a robot to perform a navigation action, is set to $\Delta t = 0.1s$. The baseline policy is trained on one scenario for total timesteps $T = 512{,}000$. In the case of cumulative training policy, the entire training timesteps are 3,072,000 because it is trained through consecutive six scenarios. Also, the maximum episode length of 128 is set for the experiments with the

first three scenarios SC1, SC2, and SC3, while the maximum episode length of 256 is used for the last three scenarios. The batch size parameter, which denotes the number of samples used for gradient batch updating of the policy model, is set using the maximum episode length parameter. The parameter settings of our best reward function are displayed in Table 6.2. This set of parameters is obtained by performing a grid search using the baseline PPO learner on three scenarios SC1, SC3, and SC5. The grid search on the parameters of reward function is considering the reaching goal reward scale, the collision penalty scale, the progress reward, and the intermediate scale while keeping other parameters fixed. The performances of such the baseline on different scenarios are collected, and the set of parameters that have the best average results on these three scenarios is selected for our remaining experiments.

Table 6.2 *The parameters of our reward function*

Parameter	Descriptions	Value
$r_{arrival}$	The reward for reaching the goal position	100
δ_d^g	The reaching goal distance threshold	0.25 m
$r_{progress}$	The reward/penalty for making progress to the goal (+ if closer, - if farther)	1.0
$r_{intermediate}$	The reward for reaching the milestone (intermediate goals between the initial and the target position)	2.0
δ_d^r	The minimum safe distance between two agents	0.6 m
δ_d^o	The minimum safe distance between an agent and the nearest obstacle	0.3 m
r_{col}	The penalty for any collision happens	-50
r_{linear}	The penalty of negative translation velocity	-2.0
$r_{angular}$	The penalty of high rotational velocity	-0.5
δ_v^a	The maximum rotational speed without getting a penalty	$\frac{\pi}{4}$

Table 6.3 and Table 6.4 display the hyperparameters of the two adapted-PPO and adapted-TRPO in our experiments. Similar to the case of reward function parameters, a

grid search is also performed to find the best hyperparameter settings of these two algorithms. In this work, the grid search is performed on the learning rate only.

Table 6.3 *The hyperparameters of the adapted-TRPO algorithm*

Parameter	Description	Value
cg_damping	The conjugate gradient damping	0.01
cg_iters	Number of conjugate gradient steps	10
ent_coeff	The entropy coefficient in the policy optimization objective	0.01
lr	The learning rate of Adam optimizer	5.5e-4
max_kl	Max Kullback-Leibler (KL) divergence between previous policy and current policy	0.001
vf_iters	Number of value function optimization mini-steps per each policy optimization step	3
λ	Advantage estimation discounting factor	0.95
γ	The discounted reward factor	0.99

Table 6.4 *The hyperparameters of the adapted-PPO algorithm*

Parameter	Description	Value
clipRange	Clipping range (a scheduling function on training time)	$0.3(1-\frac{t}{T})$
entCoeff	The entropy coefficient in the policy optimization objective	0.01
lr	The learning rate of Adam optimizer (decreasing over training time)	$7.5e^{-5}(1-\frac{t}{T})$
maxGradNorm	The maximum gradient norm clipping coefficient	0.5
nMiniBatches	The number of mini-batches training per updating	1
nOptEpochs	The number of training epochs per updating	10
vf_coef	The coefficient of the value loss function	0.5
λ	Advantage estimation discounting factor	0.95
γ	The discounted reward factor	0.99

6.1.2.2 Performance Comparison

As discussed in previous sections, there are four types of policy to be trained: two cumulative policies using TRPO and PPO, the baseline policy using TRPO for each scenario, the baseline policy using PPO for each scenario. We trained and conducted our

experiments on a machine that has an Intel Core i7 CPU and a GeForce GTX 960M GPU. The approximate time to complete 512,000 training timesteps on such a computer is around 12 to 18 hours. The mean episode reward for the cumulative training using adapted-TRPO and adapted PPO are displayed in Figure 6.4 and 6.5, correspondingly. The mean episode reward is used as the metric to detect the overfitting problem during the training process. Such a problem happens in the case of cumulative TRPO training in the last scenario. While there are a lot of fluctuating moments in the learning curse of the PPO training, the TRPO algorithm shows a smoother learning curve. This phenomenon has resulted from the effect of the "trust region" limit when updating the policy in the TRPO algorithm.

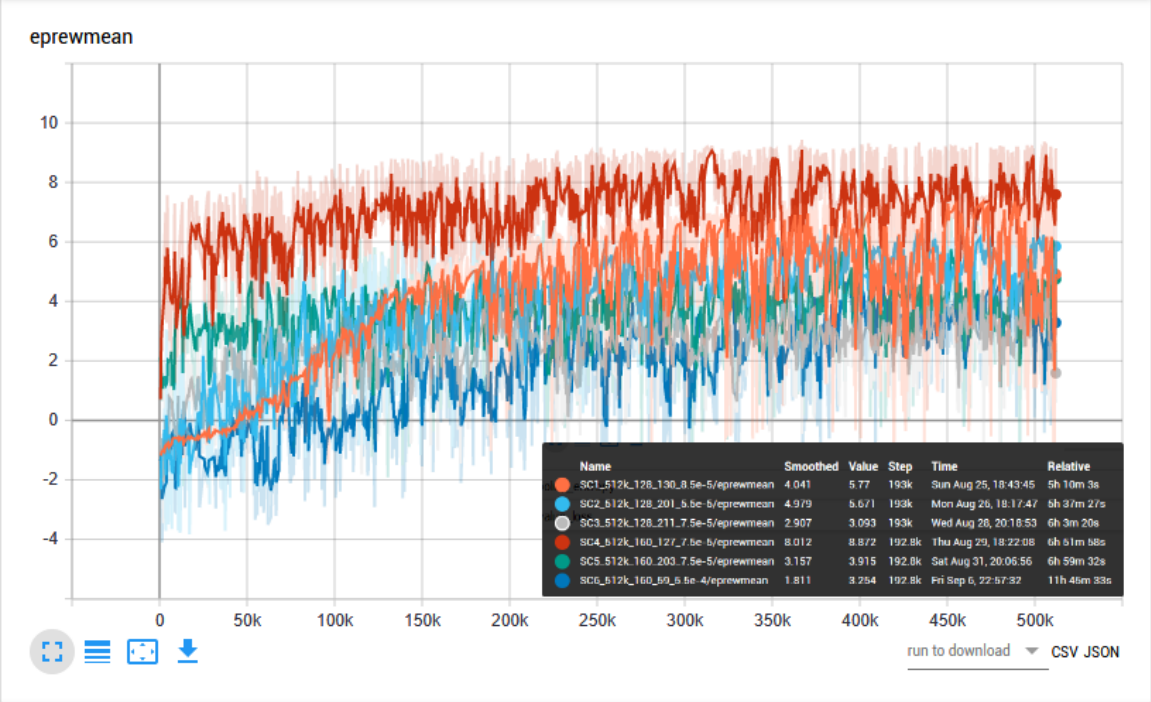


Figure 6.4 *The mean episode reward visualization of a cumulative PPO training*

Reward function. The grid search to find the best parameters of the reward function shows that the two most essential reward components are the goal-reaching reward and the collision penalty. Also, the intermediate reward can help the robots to reach

the goal in complicated cases because it gives the robots the hints to find the goal rather than performing random exploration. The other three reward or penalty components, especially the progress reward and angular penalty, are not so important. The progress reward sometimes gives the wrong direction to the robots. It happens because, in many scenarios, the direct shortest path (based on the Euclidean distance) is not available, but the robots have to take longer routes to avoid the obstacles.

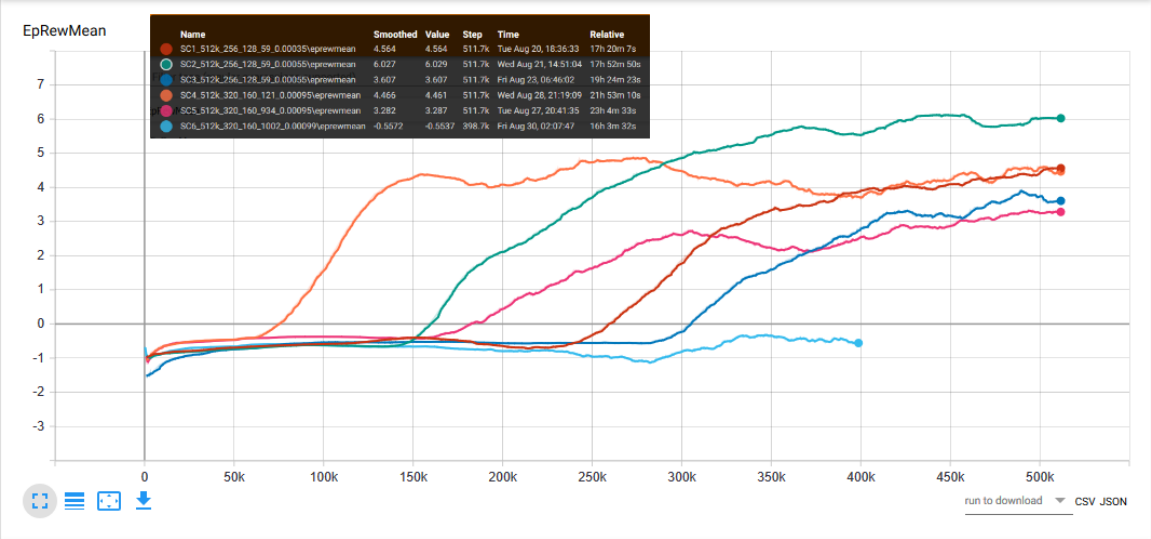


Figure 6.5 *The mean episode reward visualization of a cumulative TRPO training*

Table 6.5 *The performance comparison between our cumulative approach with baseline approaches*

Scenario	Mean Arrival Time			Success Rate		
	PPO Baseline	*Cumulative PPO*	*Cumulative TRPO*	*PPO Baseline*	*Cumulative PPO*	*Cumulative TRPO*
SC1	25.36	26.59	52.08	95%	93%	93%
SC2	25.59	40.84	42.91	96%	95%	100%
SC3	33.51	56.03	47.84	91%	78%	94%
SC4	20.17	33.26	33.81	93%	88%	96%
SC5	58.80	57.33	57.20	64%	68%	74%
SC6	77.26	47.79	67.53	69%	55%	48%

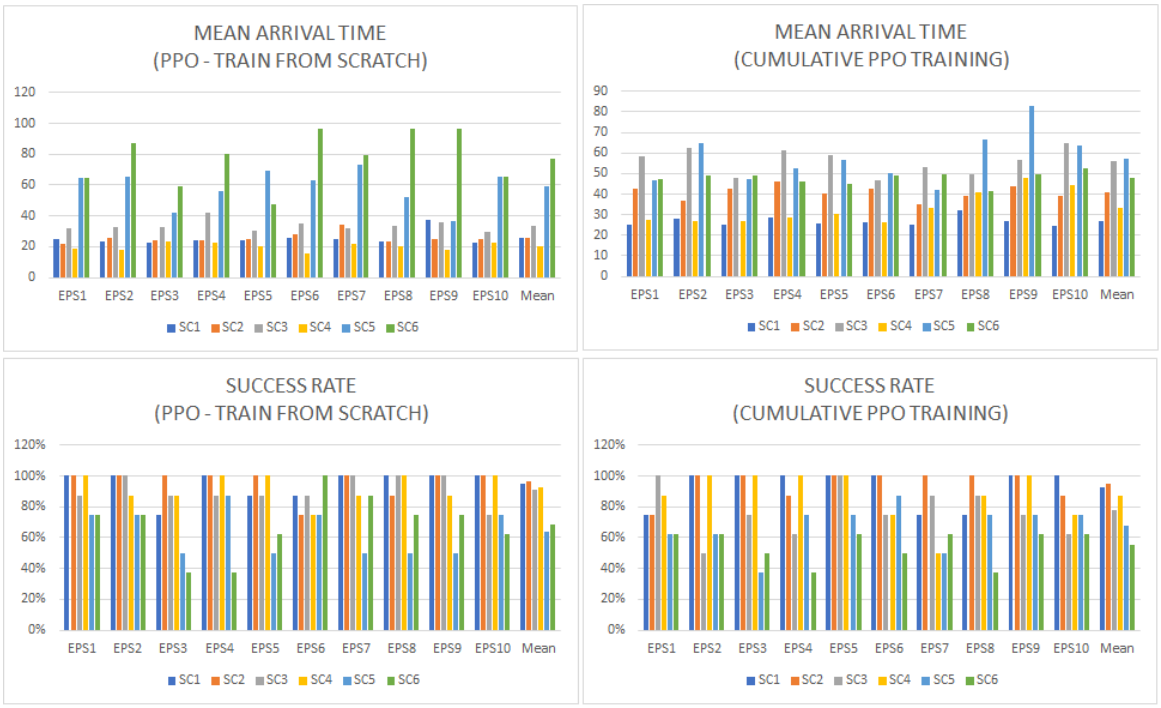


Figure 6.6 *The performance comparison between the baseline and the cumulative PPO*

Figure 6.7 *The performance comparison between the baseline and the cumulative TRPO*

Policy Performance Comparison. The performance comparison of the baseline policies with the cumulative PPO and TRPO policies are displayed in Table 6.5, Figure 6.6, and Figure 6.7. Through such the performance comparison, our cumulative training policies almost have the same performance in success rate and mean arrival time compared to the corresponding baseline policies. While the baseline policy is optimized on such a scenario and not work well if tested on other scenes, our cumulative ones show the

generalization capabilities while also keeping a stable performance in all scenarios. Among these different types of policies, the cumulative PPO training policy shows a more stable performance, even in complex situations with long navigation tasks. The policies trained by the TRPO algorithm perform well on simple scenarios with short to medium navigation tasks. However, its performance suddenly drops if they are trained in more complex situations. The recording videos of different training and testing experiments are recorded and uploaded to our public YouTube playlist[9].

This section concludes the introduction of our work on the multi-robot navigation problem. The next section introduces the second study on the multi-robot foraging problem.

6.2 Multi-robot Foraging Problems

As discussed in section 4.2.2, many promising real-world applications in swarm robotics, such as cleaning up toxins, search and rescue, and automatic construction or repairs, etc., are the instance of collective foraging tasks. The collective foraging problem can be understood as a set of robots exploring the environment, finding "food" resources, picking them up, and returning them to the "nest". The robots have limited capabilities in sensing and localization, and limited knowledge of the environments. Moreover, the swarm robotics is designed for autonomous and decentralized, so there are no centralized controller or leader.

The starting point of this research work is the two foraging algorithms that were introduced by Hoff et al. in 2010. In their algorithms, the robots used a limited range of

[9] https://www.youtube.com/playlist?list=PLbhGMGvjTex34bh2siyoJScVMcudt_9Gx.

direct communication channels to exchange two cardinality numbers, which denote the number of hops (robots) to arrive at the food/nest. Also, the robots are in one of the two following states: beacon or walker during the execution time. If a robot is in the walker state, it explores the environment to find food if it has not found one. Otherwise, it will find the nest to return food if it is carrying a food item. While moving around the environment, the walker receives the set of two cardinality numbers (food or nest) from the neighbor beacons. Then it chooses the one with the lowest value and moves to the direction of such the beacon, which sends that value. The beacon robot is always stationary. Their job is communicating with nearby beacons to update the cardinality numbers. The food/nest cardinality number is analogous to the number of hops to arrive at the nearest food source or nest. Therefore, if a beacon A is next to food/nest within a small distance, it can set its food/nest cardinality to 0. If a robot cannot sense the food/nest, it set its food/nest cardinality number as the lowest cardinality number it receives from neighbor plus one. Therefore, robot B, which gets the updated count from robot A, will set its cardinality number to 1. The authors introduced a hand-coded heuristic algorithm to help the robots to determine the role dynamically. In this research work, a deep reinforcement learning approach is studied to build a better performance policy to replace their algorithm.

Contributions. Based on Reinforcement learning as a rehearsal (RLaR) (Kraemer & Banerjee, 2016) framework, a deep reinforcement learning approach is proposed to solve the multi-robot foraging problem. The visible and hidden features have been designed for this application. Later, Deep Q-Learning (DQN) (Gu, Lillicrap, Sutskever, & Levine, 2016) policy was trained to replace the heuristic decision (Hoff, Sagoff, Wood, & Nagpal, 2010) for role choice between walker and beacon. The hidden features are available during training time, but

not available in executing time. To deal with this problem, we collected the hidden features during the experiments. We later used deep learning to train the Mixture Density Networks (MDN) (Bishop, 1994) model to generate these hidden features from the visible features. The performance confirms that our approach can substantially improve the number of food returns over time. Moreover, our approach can be generalized to other partially observable problems as well as other multi-robot learning problems.

6.2.1 Reinforcement Learning as a Rehearsal (RLaR) framework

Kraemer and Banerjee introduced Reinforcement learning as a rehearsal (RLaR) framework in 2016 for partially-observable multi-agent tasks. It is observed that while training a near-optimal policy for the robots in such a task, the restriction of the partial observability should not be strictly followed during this time. That means during training time, and generally, in a robot simulator or a laboratory setting, some hidden state features can be made available to the robots for training purposes only. Hence, the robots can learn a new action-value function $Q_\pi(s, o, a)$, in which π is the policy, s is the hidden features state, and o is the observation (the visible features to the robots all the time). The authors used the term *rehearsal features* to name such hidden features that are usually hidden but available during training time. Next, the main challenge is keeping the quality of such action-value function when the hidden features state s is not available to the robots during the execution phase.

The RLaR framework introduces an approach to marginalize the rehearsal features states in the estimation of $Q_\pi(s, o, a)$. The solution of the RLaR framework is summarized as the following two concurrent steps:

a) During the training phrase, learn a separate predictor function $f(s|o)$ to estimate the rehearsal features from the visible features

b) Using reinforcement learning algorithm to learn an optimal policy that satisfies the following equation: $\pi(o) = argmax_a \sum_s Q(s,o,a)f(s|o)$ $(Eq. 6.2a)$ during the training phase. The optimal policy will be used in the execution phase for action selection

6.2.2 Methodology

This section will present our proposed approach based on the RLaR framework to construct an optimal role section for the collective foraging problem. The main advantage of this framework is the robot agents can access the hidden features during the training phase (i.e., in a simulator or a laboratory). Still, they have to learn decision policies without relying on such hidden features to be used in the execution phase. The RLaR framework provides a hint of learning a predictor function of the hidden features from the visible features. The remaining of this section will describe as follows. First, the settings of the multi-robot foraging problem in this work are described, followed by the implementation of the baseline cardinality algorithm proposed by Hoff. Then, we present our proposed reinforcement learning approach to replace the baseline heuristic decision algorithm. Finally, the method to learn the predictor function is described.

6.2.2.1 Multi-robot Foraging Problem Settings

Problem Settings. The collective foraging problem in the scope of this work is similar to the problem setting of the central-place foraging proposed by Hecker and Moses. There are only two areas in the environment: one storing the resources for the robots to search and pickup, and another for the robots to return the resource items. Moreover, the robots in our settings are primitive, and cannot use the shared memory or modify the

environment to execute their control policies. Instead, the robots can communicate within a limited range. The cardinality-based heuristic foraging algorithm developed by Hoff et al. is the baseline algorithm of our work. According to this baseline algorithm, some robots become beacons to maintain the cardinality count to coordinate the exploration of other walker robots.

Figure 6.8 *The example of the footbot introduced by the Swarmanoid project*

Robot Design. The robots in our experiments are based on the prototype of the footbot[10] robot. This one was developed in the Swarmanoid project (Figure 6.8). That is a two-wheeled modular robot with a rich set of sensors and actuators, including a range and bearing module, a distance scanner, a vision module, and others. The selection of this kind

[10] Swarmanoid project: http://www.swarmanoid.org/swarmanoid_hardware.php.html

of robot prototype is matching with the assumptions of the baseline Hoff et al.'s algorithm as the following:

- the robots operate in differentiate driven kinetics mode, so they move and turn in a continuous space
- the robots use the range and bearing sensors and actuators with the maximum distance of 1.5m to directional communicate with the neighbors
- the robots can only sense the food/nest and other obstacles in the proximity of them

In our problem settings, the range-and-bearing sensors provide the communication channel between agents. This channel has a maximum message length of 10 bytes. The communication message consists of the id of the sender robot, state of the sender (beacon or walker), and food/nest cardinality number. This message content is packed to match the 10 bytes length requirement. Moreover, when a robot receives a message from a neighbor, it also knows the relative angle and approximate distance of the sender in a precise degree accuracy. Therefore, it can move closer to the sender robot in quick and high efficiency.

6.2.2.2 Implementation of the Baseline Foraging Algorithm

In this work, we implemented the cardinality foraging algorithm of Hoff et al. as the baseline for performance comparison with our approach. The robots operate in one of the following two roles: beacon and walker. The walker robots are always in the food searching state or nest searching state. When the walker robots haven't found the food source location, they explore the environment to find and pick up the food items. On the other hand, if they already carry a food item, they search for the nest to return the food. Because the robots are designed with limited knowledge of the environment, this searching process bases on the random walk without any hint. The foraging algorithm improves this

process by introducing the role of the beacon. The beacon robots are stationary and maintain the food cardinality and nest cardinality. Food cardinality denotes the number of hops (beacon robots) to reach to nearest food location from itself.

Similarly, the nest cardinality is designed for the nest. In the best case, there is at least one shortest path formed by the fixed beacons from the food to the nest. Such an example is shown in Figure 6.9. Here, the set of beacon robots $< b_1, b_2, b_3, b_4, b_5 >$ forms the shortest path from the food to the nest. The food/nest cardinalities are drawn beside the shape of the beacon robots. It is easy to see that the food cardinality is increasing from 0 (the beacon b_1 next to the food) to the beacon b_4 (4) next to the nest, while the nest cardinality is reversed. The group of walker w_i is denoted by two colors: blue is the food searching ones and green for the nest searching ones.

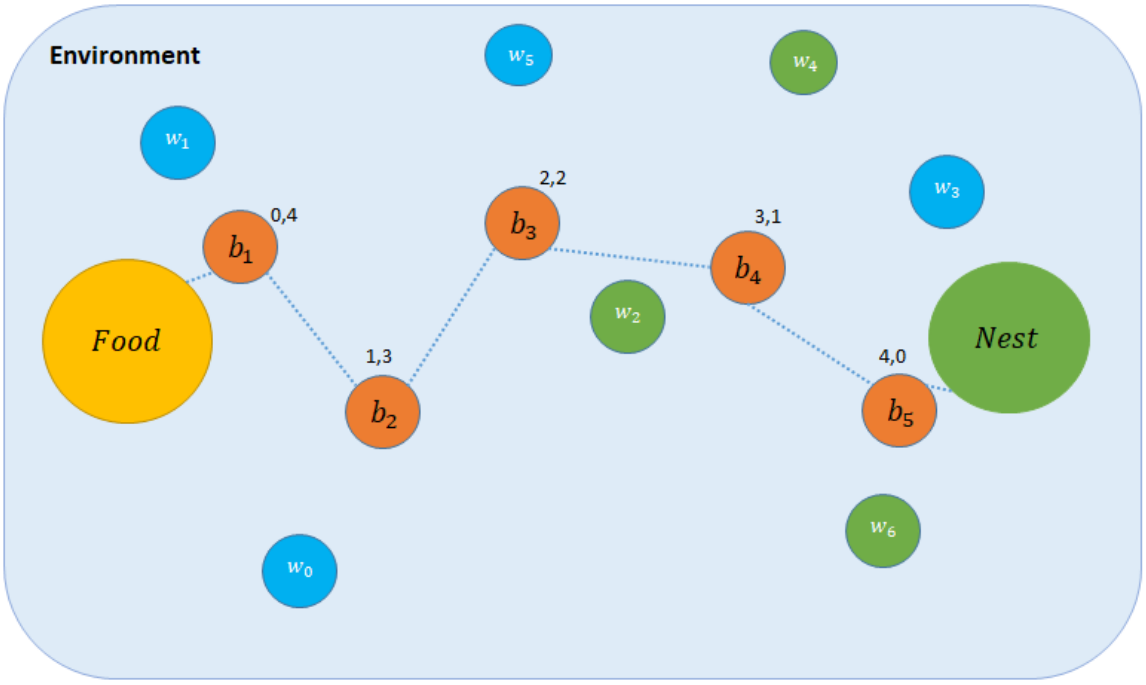


Figure 6.9 *The conceptual idea of the baseline foraging algorithm*

Our rule of updating the food/nest cardinality is implemented on the description in Hoff et al. ‘s algorithm with a small modification to match with our problem settings. Initially, the food/nest cardinalities of all the beacon robots are set to ∞, which means there is no information on the nearest food or nest. If a beacon robot has a small enough distance that can sense food or nest, it updates its food or nest cardinality to 0. At a timestep, each beacon receives messages from the neighbor beacons. The message content consists of the sender ID, state, and the food/nest cardinalities. Next, each beacon will compare its food/nest cardinalities with the received ones. If its food/nest cardinalities are not 0, it updates its values to the minimum received food/nest cardinalities plus one. That is the way to form the shortest path from the food to the nest. Finally, the beacon robots will broadcast their new messages, including their ID, states, and their food/nest cardinalities, to neighbors. The walker robots only receive messages from neighboring beacons. If they are in the food searching mode, they attempt to move closer to the beacon with the smallest food cardinality and similar for the nest searching mode. However, if the food/nest cardinality is ∞, the walker robots will execute the random walk behavior to explore or avoid the obstacles. In our experiments, there are no static obstacles; only other robots are considered as dynamic obstacles.

The baseline foraging approach of Hoff et al. proposes a hand-coded heuristic algorithm to determine whether to keep or change the current role. If a beacon receives messages from less than two beacons, it decides to become a beacon. Also, if a beacon receives signals from three or more beacons, it chooses to become a walker with the probability of p. In our baseline experiment, we used the same value $p = 0.3$, as suggested in the original paper. In this work, the above heuristic decision function is replaced by a

reinforcement learning policy. This policy maps directly from the visible features and the predicted hidden features (by the predictor function) to the role for the next timestep.

Simulation Environment. In this research work, we use the ARGoS simulator (Pinciroli et al., 2012)[11], which is another product developed by the Swarmanoid project, for the simulation of our experiments. The visualization of our collective foraging settings is displayed in Figure 6.10. For performance comparison, we try to keep the same settings in the original paper, which are the same *communication radius* and *food-nest separation distance*. However, we increase the world size to $5m \times 9m$. Also, since our ball-like footbot has a radius size of roughly $0.18m$, we limit the number of robots in the environment to 40 to keep the same *robot density*. Moreover, we set the maximum number timesteps of 3,000 for each episode, instead of 750 (as in the original paper).

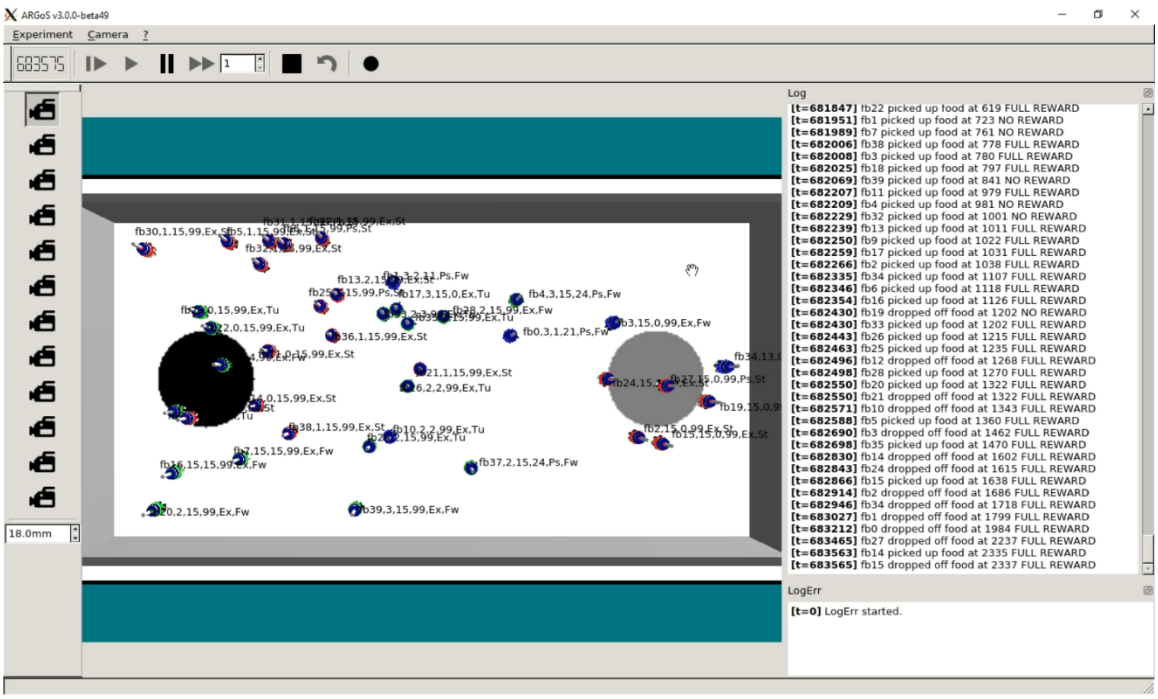


Figure 6.10 *The GUI of the ARGoS robot simulator*

[11] ARGoS homepage: https://www.argos-sim.info/index.php

The number of food returned is used as the only performance metric, not including the energy (cost). It is not easy to evaluate the performance based on the energy because of the differences in the problem settings of our approach and Hoff et al.'s work. The number of food returned metric is more relevant and stable optimization objective for our method.

6.2.2.3 MARL approach for the Collective Foraging

It is a common belief that a multi-agent reinforcement learning framework is used for multi-agent systems such as swam robotics. Also, in a multi-agent system, there are different roles or specializations, which have distinct control mechanisms. For such a system, the multi-agent learning framework can be applied by creating the equivalent agents and simultaneously learning the optimal control behaviors of them. However, for the swarm robotics system that can have a large number of robots, this kind of specializations is not recommended. The robots in the swarm should have the same control mechanism to provide the fault tolerance properties of such the swarm. Then, such different roles or specializations should be embedded in the control mechanism and should be chosen dynamically. Also, we need to consider the joint action spaces between multiple agents in the multi-agent learning design. This requirement makes the design, hardware, and software requirements more complex. Therefore, an adapted, single, reinforcement learning approach for swarm robotics is more beneficial.

Since all robot agents in the swarm use a single controller (policy), we apply the "*centralized learning and decentralized execution*" scheme to train such a policy. In this scheme, the agents (the actors), which receive the same copy of the current policy. Then each agent uses that policy to generate the actions, interacting with the environment, and

collect the new experiences. Experiences are collected from all agents and feed to the single learner to update the policy during the training phase. After training, each robot receives the final copy of the policy and execute it on its own without synchronizing with the central learner.

Observation features. The robots in our system can send/receive messages within a certain distance. As discussed in the previous section, the message content consists of the following information: sender ID, sender state, and food/nest cardinality. Based on this information received with the internal state of each robot, we design the set of visible features and hidden features that are presented in Table 6.6.

Table 6.6 *The descriptions of visible features and hidden features*

Type	Symbol	Descriptions
Visible	c_s	The current role of the robot, (0: walker and 1: beacon)
	h_f	Carrying food or not (0: false and 1: true)
	n_b	Number of neighboring beacons
	n_w	Number of neighboring walkers
	d_{fb}	Distance to the farthest beacon
	d_{nb}	Distance to the nearest beacon
	f_{fb}	Food cardinality of the farthest beacon
	n_{fb}	Nest cardinality of the farthest beacon
	f_{nb}	Food cardinality of the nearest beacon
	n_{nb}	Nest cardinality of the nearest beacon
Hidden	d_f	Distance to the food source
	d_n	Distance to the nest

In the set of visible features, the c_s feature denotes the current role of the robot (0: walker, and 1: beacon). Also, the h_f feature indicates whether the current robot is carrying food or not. The remaining visible features are easy to calculate from the reived messages from the neighboring robots. In our experiment, the walker robots also send the messages, but their food/nest cardinalities are set to ∞ (because they do not maintain such number).

The two hidden (rehearsal) features, which are the distances to the food or nest, usually are not available to the robots but can be provided easily by the simulator during the training phase. In the experiments, for each of the visible and hidden features, except the first two visible features, we divided it with the corresponding maximum value to normalize it into the range of (0,1). For instance, for distance features, the maximum distance is 45m because our environment is bounded by $5m \times 9m$. Similarly, because the cardinality can not larger than the number of robots in our experiments, we used that number to represent the infinity cardinality.

Figure 6.11 *The example of neighboring image feature (10 times zoomed-in)*

Besides those numerical features, we also build an image-based feature from the messages received from the neighboring robots. We call this kind of feature as neighboring image. With the observation that the messages are received from neighboring robots in the radius of 2m, we use a 48x24 grayscale image. Figure 6.11 displays an example of this kind of feature. This image is a concatenation of two sub-images: the left one represents the food cardinalities of neighboring robots, and similarly, the right one represents the nest

cardinalities. So, one of such sub-image is mapped to the 2m rectangle boundary of the communication radius of a robot. While the grayscale values from 200 to 255 represent the background (no occupied by the robots), the grayscale from 0 to 200 will be mapped to the food/nest cardinality monotonically. That means the 0 food/nest cardinality is mapped to 0 grayscale value, and the infinity one will be mapped to the grayscale value of 199.

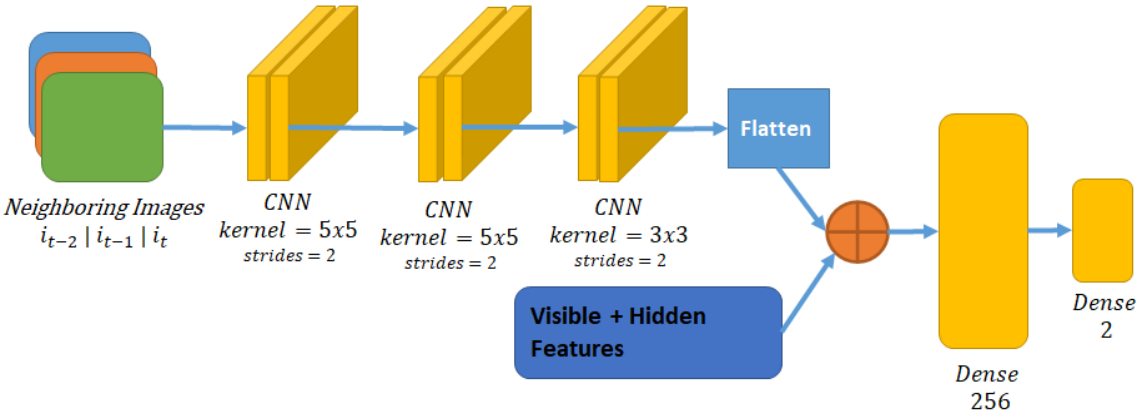


Figure 6.12 *The neural network architecture of the policy model*

Learning algorithm. In this research work, Deep Q-Network (DQN) is utilized to learn the optimal policy from the set of visible features, hidden features, and neighboring image features. The mathematical model and description of the DQN algorithm are explained in section 2.6. Moreover, in our experiments, we applied two improved techniques for DQN, which as the fixed Q-targets and the Prioritized Experience Replay (PER) (Schaul, Quan, Antonoglou, & Silver, 2015). Our neural network policy model contains three CNN layers, a flatten layer, a concatenation layer, and two Dense layers. Figure 6.12 presents the architecture of this policy model. In this research work, to learn better from the past observations of the neighboring robots, we stacked the last three neighboring images to make the image input $[i_{t-2}, i_{t-1}, i_t]$. This input is fed to the three CNN layers to produce the latent output. This latent output is flattened at the next layer and

concatenated with the last three numerical features (visible and hidden features) to feed to the last two Dense layers. The last Dense layer has the size of two because we have only two actions (walker or beacon). From these two outputs, the one that has a higher probability is selected as the best action (the role of the robot).

6.2.2.4 Learning the Predictor Function

To estimate the $\sum_s Q(s,o,a)f(s|o)$ in the equation (Eq. 6.2a), a viable solution is learning a function approximation, such as a neural network model. Such a model could be trained to capture the distribution f and map the visible features o to the rehearsal features s. Then, this model could be utilized to estimate the value of s when only visible features o are available, such as in the execution phase. However, this estimation is not accurate because $\sum_s Q(s,o,a)f(s|o) \neq Q(\sum_s sf(s|o),o,a)$ whenever f is not a degenerate or deterministic distribution. That requires us to learn directly the probability distribution $f(s,o)$.

In this work, the main insight is that $f(s|o)$ is a non-degenerate distribution. Therefore, we make use of the Mixture Density Network (MDN) (Bishop, 1994) to train a Gaussian mixture model (GMM) to approximate that distribution. We use a similar neural network as the policy model in Figure 6.12 with some modifications to present the MDN model. Compared with the policy model, the differences are a) instead of using the last three neighboring images, the current image is used as the input only, b) the last six visible features are used as the numerical input features c) the last Dense layer has the size of $3 \times m$, in which m is the number of GMM components. Each component in the GMM model has three elements: the component weight ϕ, the component mean μ, and the component variance σ.

6.2.3 Experimental Work

As described in the previous section, we used ten visible features, two hidden/rehearsal features, and the stack of last three neighboring images to learn an optimal policy to select the role between walker and beacon for the next timestep. To avoid the problem of too much role changing in a short time, we applied a persistent count of 20 timesteps for a decision. In our experiments, the following reward structure was used:

a. +1 if a robot pick up a food item;

b. +10 if a robot return a food item to the nest;

c. -0.5 when a robot changes its role

d. -0.1 when a robot keeps its role

In our experiments, to discourage the behaviors of picking up or returning food by the random walk, we did not reward the agents in such cases. The robots are rewarded for any pickup/return in the state of pursuing a target. Moreover, to encourage the cooperation between the beacons and the walkers, we applied the reward sharing ratio of 9:1 between the walker and the target beacon, which guide the walker to successfully pickup/return food items.

To obtain the initial training dataset for the predictor function of the rehearsal features, we ran the hand-coded baseline policy for 100 episodes to collect 600,000 samples. Later, we performed offline MDN training with the following best set of hyperparameters: learning rate = 0.0001, number of GMM components = 2, batch size = 256, number of training epochs = 500, kernel regularization penalty = 0.001, and drop out rate = 0.25. The above set of parameters is the result of a grid search process. The early stopping technique was also used to prevent overfitting by periodically evaluating the

trained model on an unseen validation dataset. The RMSE of the best MDN model is 0.1133.

An episode in our experiments has the maximum length of 3,000 timesteps, but each robot only makes 150 role decisions (note that we used the persistent count of 20 timesteps of choice). After the end of each episode, we reset the starting locations of all robots randomly. During policy learning by the DQN algorithm, we applied a modified ϵ-greedy exploration strategy. In this strategy, the epsilon is exponentially dropped from 1.0 to 0.02. The value epsilon ϵ controls the action selection in the learning process similar to the typical ϵ-greedy algorithm, in which random action is selected with a probability ϵ. However, the action generated from the policy is chosen with the remaining probability $1 - \epsilon$. Also, we used the DQN learning is executed with the following settings: learning rate decay from the starting value of 0.0001, discounted factor = 0.99, replay buffer size = 200,000. Also, we used the default parameters of the Prioritized Experience Replay introduced in the original paper.

The number of returned food in each of the 3,000 episodes are collected. For comparison, we collected such the results using four different policies: a) the hand-coded policy, b) the DQN learning without rehearsal features, only using ten visible features and the neighboring image features, c) the DQN learning with all features, and d) the DQN learning with all features, except the rehearsal features are replaced by the predicted values from pre-trained MDN model. The number of return food are smoothed using the exponential moving average technique (windows size of 100 episodes, mixing rate of 0.01) and the cumulative average method. Later, the results are averaged over five independent runs and plotted using the Gnuplot library in Figure 6.13. The three kinds of DQN

experiments are executed with the same settings and parameters. Since all the DQN learning experiments start with the uniformly random policy, the beginning of their plots is the upper bound of the random policy performance. Figure 6.13 also shows that the hand-coded policy is better than the random policy with a statistically significant difference.

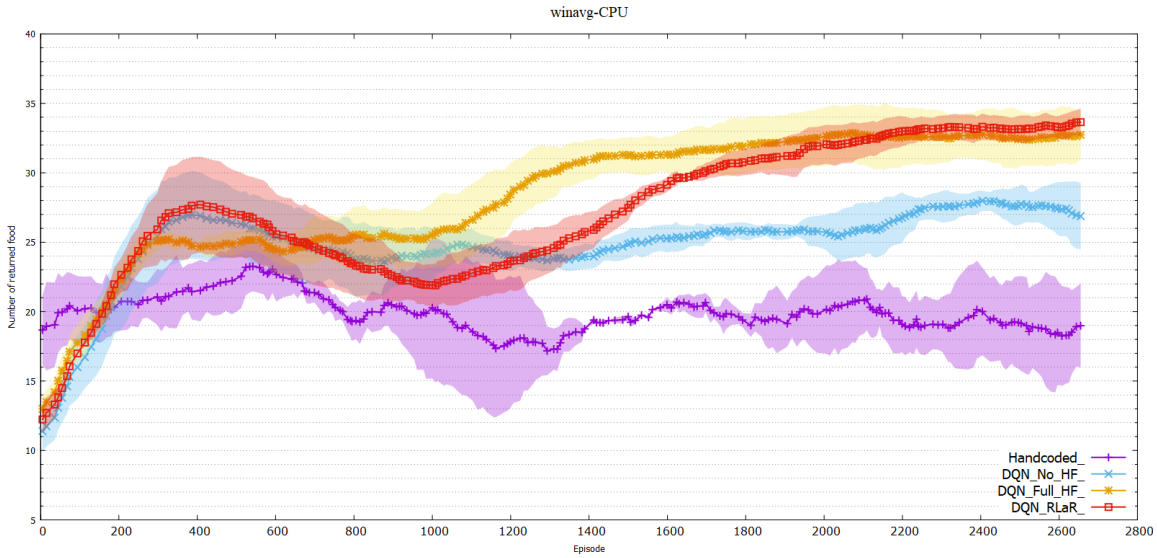


Figure 6.13 *The moving window average learning curves of the number of returned food of four different DQN policies vs. the baseline hand-coded policy*

The plotted results of the number of return food are displayed in Figure 6.13 (window moving average) and Figure 6.14 (cumulative moving average). In these plots, the baseline hand-coded learning strategy is marked with a violet color. The DQN learning without using hidden features, called "*DQN_NoHF*," is drawn by the light blue color. Next, the DQN learning using the original hidden features, called "*DQN_FullHF*," is drawn with the yellow color. Finally, the last DQN learning using the MDN predicted hidden features, called "*DQN_RLaR*," is drawn with the red color. All DQN learning approaches are performed more effectively than the implementation of the baseline hand-coded policy. Both the methods using the hidden features learn better than the learning without hidden

features, which are the main benefits of the RLaR framework. They also appear to asymptote at a slightly better policy than the DQN without hidden features.

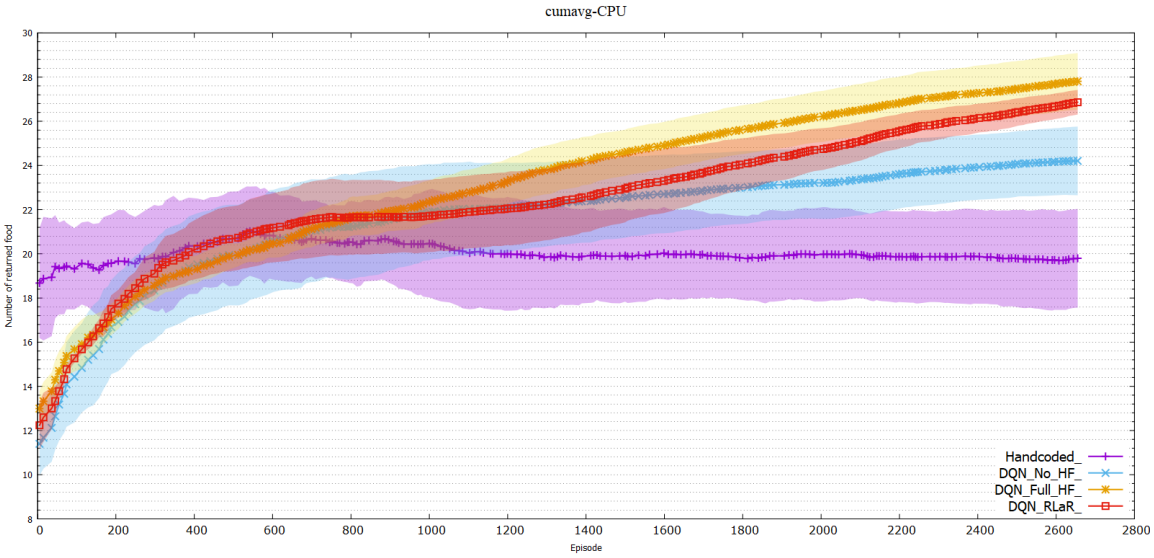


Figure 6.14 *The cumulative window average learning curves of the number of returned food of four different DQN policies vs. the baseline hand-coded policy*

Both the moving window average and cumulative average plots show that the baseline has a stable average performance throughout the evaluation time. All the DQN learning approaches start with deficient performance because of the effect of the random policy at the beginning of the learning process (because of the high epsilon in our modified ϵ-greedy exploration schedule). The "*DQN_FullHF*" approach performs best and has an asymptotic learning behavior as expected because of the benefits from the hidden features – knowing the real distance to the food/nest. The gap performance between this strategy with the "*DQN_NoHF*" reflects the essential features. The plots show that our RLaR framework can fix such a gap, even in the case the hidden features are not available for the robots out of the simulation, although there is a little drop in the performance compared to the "*DQN_FullHF*" approach.

Another significant observation is that our RLaR approach shows less variance compared to the DQN learning approach without using hidden features and the heuristic baseline algorithm. Variance reduction is believed as another benefit of the RLaR framework (Kraemer & Banerjee, 2016). That is because the action-value function learned only from the visible features o, fails to account for the *aliasing* phenomenon, caused by the lack of the informative hidden states s. By contrast, RLaR explicitly makes use of the hidden features to learn the value function and later marginalizes them out using the predictor function. That is an example of the *probabilistic anti-aliasing* capabilities of the RLaR framework. In addition to the visible features, RLaR can use some insight into the hidden features via the predictor function, even when the hidden features are no longer available. Therefore, the performance of RLaR is less subject to the variability resulting from the lack of any understanding of the hidden state at any given step.

6.3 Discussions

This section concludes the chapter on deep reinforcement learning approaches to solve collective task learning problems in swarm robotics. This chapter has shown two adapted deep reinforcement learning to address two crucial problems in the swarm robotics: navigation and collective foraging. Both approaches have been designed to make use of the local observations and avoid the dependency on communication between agents. This kind of design, in our belief, is more robust in real-world applications.

CHAPTER VII – CONCLUSIONS

This chapter summarizes the main contributions of this work, recognizes promising applications, and describes future research directions.

7.1 Findings and Limitations

There are five studies have been introduced in this thesis. While the first three studies are related to information diffusion modeling and predicting in social networks, the last two studies are performed on collective tasks learning in the swarm robotics domain. In the first study, a multivariate time series analysis is used for information diffusion modeling on social networks. We also propose time series clustering methods with DTW as distance measurement for understanding patterns of information diffusion processes. Later, a useful prediction model based on LSTM is introduced to predict different features of information diffusion on social networks in future timelines. The next study takes a deeper step to model the user's influence on social networks. Our proposed method based on deep learning is used to capture the global influences of nodes over time and predicting the temporal volume of information diffusion processes. Multi-Feed Weighted Topic Embeddings (MFWTE) model is introduced in our third research to analyze user network interactions and patterns of topic diffusion on Twitter. The experimental results show that our model outperforms sing-feed models in multiple tasks such as friend recommendations and retweet behavior prediction and help to understand different topic diffusion patterns of various tiers of users.

In the domain of swarm robotics, a new cumulative training approach using transfer learning is introduced in our fourth study to build a multi-robot collision-free navigation system. The performance shows that our method can create a more generalized policy that

can work well with complex indoor environments (with short, medium, and long navigation tasks). Lastly, a deep reinforcement learning approach, which is based on Reinforcement learning as a rehearsal (RLaR) framework, is proposed to solve the multi-robot foraging problem. Our approach makes use of the rehearsal (hidden) features that are only available during the training phase (or laboratory settings) by learning a predictor function. Such a predictor function is learned by the Mixture Density Network. The performance evaluation has demonstrated the effectiveness of our approach to increase the number of returned food. Moreover, this approach can be generalized to other tasks in swarm robotics.

7.2 Applications

Some practical applications can benefit from different studies that are presented in chapter 5 and chapter 6. This section briefly describes the general application areas of this work.

7.2.1 Social network analysis applications

The increasing usage of social networks provides us an excellent chance to study information propaganda phenomena. This work contributes thorough research about analyzing and predicting how information spreads over social networks. Therefore, it can be applied to different real-world applications. Such two most promising applications are an end-to-end real-time application of information diffusion analysis over the social networks and influence analysis applications. The real-time information analysis applications can be built by following the general framework of our first study. Such a framework covers the collection of real-time users' tweets from social networks, data preprocessing, identifying the diffusion patterns, and finally predicting the diffusion rate in the future. Next, meaningful real-time visualization graphs are embedded on the front-

end of the applications to provide up-to-date and useful information to the end-users. Politics and promotion campaigns can be the two most promising targets for our applications. During political campaigns, such applications can become very popular because they capture the hot topics and, importantly, demonstrate real-time statistics to understand the current picture of the ongoing elections. For promotion campaigns, such applications can also be used to understand the popularity and potential customers' feelings of some products.

Our methodology can provide a real-time analysis to study and maximize the social influence on a topic. Following our proposed influence and topic diffusion models, we can identify different factors that affect the diffusion patterns and global diffusion rate of a topic. Later, such identified factors can be used to embed into an influence maximization model an optimize to maximize the influences. This method is best for promotion campaign or viral marketing applications. Our studies have pointed out different factors such as retweet/replies, number of friends or followers of involved users, influence characteristics of users, etc. that can affect the diffusion rate of a topic. By collecting historical data or initial data, and using our proposed prediction models, we can sketch out different scenarios that can happen in the future. Finally, we can select the best scenarios in which the influence is maximized for the target topics.

7.2.2 Swarm robotics application

The application of swarm robotics is increasing more nowadays. Therefore, swarm robotics research is an active research domain that involves different disciplines. Collective behavior, which inspired by the society of animals, is one of the hot research topics because of many real-world applications benefits. This work has contributed throughout research

on accelerating the learning capabilities of such behaviors of swarm robotics. In the multi-robot navigation study, we have demonstrated the promising application of using swarm robotics in complicated indoor settings such as apartments, buildings, warehouses, and factories, etc. Navigation is an essential task of swarm robotics. Therefore, a robust approach should use fewer resources and avoid communication as much as possible. Our approach relies only on local observations from sensors to build an effective navigation system. The second study strengthens this approach by introducing the benefits of the RLaR framework, which helps the robots make use of hidden features even after the training phase. Such methods can be generally applied to other tasks of swarm robotics.

7.3 Future Work

In the conclusion of this thesis, there are some essential aspects of future works that can be highlighted in the following subsections.

7.3.1 Social network analysis

In social networks, influence analysis and maximization are still one of the fundamental research topics analysis because of the benefits to real-world applications, especially political/promotion campaigns, or viral marketing. Nevertheless, topic modeling, community detection, and network structure analysis are also received a lot of attention. With the massive data availability of social networks, deep learning is expected to be the core research methodology to study user network interaction on social networks, analyze the influence, maximize the impact and predict the information diffusion processes. Furthermore, numerous social network analysis applications such as bot detection, friends or topics recommendation, and community detection for political/promoting campaigns will emerge in this domain.

7.3.2 Swarm robotics

Common problems of Deep Reinforcement Learning (DRL) methods are the training difficulty and the slow convergence rate. Research has shown that difficulty in environment exploration and inefficient sampling of experiences are the main reasons for these problems of DRL. Even this problem may be worse in the swarm robotics domain because of the differences between robot simulator and the real ones. Therefore, more research should be done to improve the learning efficiency in swarm robotics in both simulation and real platforms. With the expectation that the hardware and software of the robotics platform will be enhanced more in the future, the more deep reinforcement learning applications in this domain will come.

In the context of swarm robotics, multi-tasking is a widespread phenomenon. Our second study of collective foraging is an instance of this kind of problem. In the multi-robot foraging, there are only two roles: walker and beacon. However, in real-world applications, there may be more of such similar roles or specialization depending on the type of applications. Therefore, one of the future research objectives is proposing an effective multi-agent multi-task learning scheme for all robot agents in the swarm. Hierarchical reinforcement learning (HRL) approaches can be the answer to such a problem. These approaches are close to what humans learn and behave. In general, HRL is expected to solve many complex tasks because of the better generalization, more scaling up (efficient learning), and task abstraction.

APPENDIX A – Information Diffusion Time Series Clustering Performance

Table A.1 to A.10 represents the clustering performance (CVIs) of different features in our multi-variate information diffusion time-series dataset (see section 5.1 for more details).

Table A.1 *Cluster performance of #**tweet** feature with different number of clusters*

Note: the values of D have been removed because of invalid values

CVIs	4	5	6	7	8	9	10
Sil	0.007	0.004	0.068	0.024	0.041	0.048	-0.004
SF	0.005	0.0004	0.011	0.006	0.009	0.004	0.004
CH	62.15	46.91	193.24	102.76	0.013	0.019	0.17
DB	2.56	6.43	1.88	8.44	45.48	11.88	2.32
DBstar	7.70	2.58	2.37	9.97	15.02	20.37	32.83
COP	19.02	61.02	8.68	60.88	43.89	66.86	45.85

Table A.2 *Cluster performance of #**retweet** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	-0.1413	-0.1315	-0.0953	-0.1366	-0.0994	-0.1283	-0.1275
SF	0.0410	0.0189	0.0017	0.0036	0.0038	0.0098	0.0007
CH	39.004	38.983	55.7518	52.7244	56.3037	33.9213	124.678
DB	1.6294	1.5824	1.5051	1.5907	1.5201	1.5301	1.7140
DBstar	1.9392	1.9640	1.8867	1.9991	2.809	2.2354	2.9524
D	0.026	0.026	0.002	0.0105	0.0105	0.0177	0.0119
COP	1.0800	1.0656	0.8991	1.0337	0.9104	1.0603	0.8111

Table A.3 *Cluster performance of #**direct_influence** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.13	0.11	0.09	0.02	0.01	0.03	0.05
SF	0.02	0.01	0.01	0	0.01	0.01	0
CH	286.13	149.05	52.38	193.69	244.37	139.23	147.64
DB	1.47	2.19	5.21	8.25	23.56	7.24	7.49
DBstar	1.53	4.28	2.15	9.29	3.13	4.52	3.63
D	0.001	0.02	0.05	0.12	0.98	0.34	0.03
COP	0.9	1.47	2.66	1.39	2.69	1.85	2.43

Table A.4 *Cluster performance of #**indirect_influence** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.07	0.03	0.04	0.03	0.03	0.05	0.02
SF	0.02	0.01	0.01	0	0.01	0.01	0
CH	214.94	79.54	183.28	93.59	124.17	114.32	127.14
DB	1.11	1.02	0.91	0.95	1.06	7.24	7.49
DBstar	1.38	2.18	2.77	1.97	2.23	2.89	1.59
D	$1.1e^{-16}$	$2.1e^{-13}$	$5.1e^{-12}$	$7.1e^{-12}$	$1.1e^{-11}$	$2.4e^{-11}$	$1.3e^{-10}$
COP	0.4	0.91	1.22	1.56	1.45	0.72	2.66

Table A.5 *Cluster performance of #**negative_percentage** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.603	0.6053	0.6213	0.6125	0.6191	0.621	0.6074
SF	0.0774	0.0736	0.0316	0.0376	0.0192	0.0165	0.0712
CH	1173.613	880.933	389.815	580.705	498.876	437.810	705.258
DB	1.1849	1.1021	1.0448	1.0928	1.1736	1.1668	1.0562
DBstar	1.4971	1.3908	1.3182	1.4595	1.4987	1.4281	1.3570
COP	0.3691	0.3686	0.3232	0.3670	0.3246	0.3236	0.3681

Table A.6 *Cluster performance of #**neutral_percentage** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.01	0.01	0.02	0.01	0.01	0.01	0.01
SF	$3.78e^{-11}$	$2.16e^{-10}$	$7.41e^{-9}$	$9.66e^{-9}$	$1.3e^{-11}$	$9.05e^{-14}$	$4.06e^{-14}$
CH	$1.4e^{3}$	$1.15e^{3}$	$1.01e^{3}$	$8.57e^{2}$	$7.46e^{2}$	$6.58e^{2}$	$5.41e^{2}$
DB	1.55	1.55	1.39	1.42	3.63	3.98	3.71
DBstar	2.35	2.31	1.86	1.97	4.38	5.14	4.94
COP	0.44	0.41	0.36	0.37	0.45	0.45	0.44

Table A.7 *Cluster performance of #**positive_percentage** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	5.81	5.83	6.89	5.83	5.8	5.92	5.9
SF	1072.28	0.0561	7343.47	3331.55	3293.53	1152.34	2866.21
CH	986.03	897.18	255.73	571.19	476.55	326.23	286.91
DB	1.8681	1.6435	1.3867	1.6013	1.7256	1.8939	1.4362
DBstar	1.9015	2.2819	1.6278	2.2459	1.9643	2.5989	2.6557
COP	5.24	4.63	3.51	3.97	3.967	3.52	3.52

Table A.8 *Cluster performance of #**negative_average_score** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.9134	0.8729	0.8670	0.8638	0.8639	0.8659	0.8754
SF	0.2454	0.0967	0.0673	0.0546	0.0794	0.0708	0.063
CH	2.49	312.69	250.42	208.9	179.03	156.76	199.36
DB	1.0005	1.9551	1.8339	1.7003	1.5903	1.5151	1.1768
DBstar	1.0919	2.2231	2.1659	2.138	2.1413	2.0666	1.4107
D	0.2215	0.0381	0.0381	0.03819	0.0381	0.0393	0.085
COP	0.476	0.9332	0.9327	0.9322	0.9319	0.9307	0.5248

Table A.9 *Cluster performance of #**neutral_average_score** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.2708	0.1808	0.1666	0.0293	0.0124	0.0391	-0.0544
SF	$1.3e^{-7}$	$8.7e^{-9}$	$5.83e^{-11}$	$1.16e^{-11}$	$3.3e^{-14}$	$1.77e^{-15}$	0
CH	120.85	349.05	752.13	513.69	444.37	389.38	347.64
DB	7.2	7.29	5.11	5.25	7.56	7.74	7.79
DBstar	8.38	8.58	5.55	5.69	9.11	9.57	9.76
COP	0.4	0.37	0.36	0.39	0.39	0.45	0.44

Table A.10 *Cluster performance of #**positive_average_score** feature with different number of clusters*

CVIs	4	5	6	7	8	9	10
Sil	0.02	0.03	0.04	0.02	0.02	0.01	0.01
SF	0.01	0.01	0.02	0.01	0.01	0.01	0.01
CH	67.37	374.56	540.15	234.34	458.59	401.44	357.13
DB	2.601	1.8376	1.7582	1.9375	2.628	2.5566	2.4795
DBstar	1.8914	2.0024	1.987	1.8369	1.8803	1.8183	1.7748
COP	0.6119	0.6104	0.0607	0.6067	0.6001	0.5796	0.5792

APPENDIX B – Performance Comparison of Different Topic Embeddings Models

Table B.1 to B.5 displays the performance comparison between three topic embeddings models for the three tasks: user friendship recommendation, retweet link prediction, and topic diffusion patterns analysis.

Table B.1 *The performance of SVM on friendship recommendation task using different feed types*

Embeddings Types	Precision		Recall		F1 score		Accuracy
	0	1	0	1	0	1	
Single-Feed (Tier 1)	0.62	0.39	0.59	0.58	0.56	0.47	0.57
Multi-Feed (Tier 1)	0.69	0.75	0.78	0.65	0.73	0.70	0.72
Single-Feed (Tier 2)	0.58	0.54	0.36	0.75	0.43	0.62	0.55
Multi-Feed (Tier 2)	0.68	0.72	0.75	0.65	0.71	0.68	0.70
Single-Feed (Tier 3)	0.73	0.66	0.55	0.82	0.61	0.72	0.68
Multi-Feed (Tier 3)	0.83	0.88	0.89	0.81	0.86	0.85	0.85

Table B.2 *The performance of SVM on retweet prediction task using different feed types*

Embeddings Types	Precision		Recall		F1 score		Accuracy
	0	1	0	1	0	1	
Retweeted View (Tier 1)	0.10	0.44	0.17	0.72	0.11	0.50	0.41
Single-Feed (Tier 1)	0.51	0.54	0.17	0.88	0.25	0.66	0.56
Multi-Feed (Tier 1)	0.59	0.60	0.49	0.68	0.51	0.62	0.58
Retweeted View (Tier 2)	0.52	0.02	0.92	0.05	0.62	0.03	0.48
Single-Feed (Tier 2)	0.51	0.02	0.92	0.03	0.65	0.03	0.50
Multi-Feed (Tier 2)	0.55	0.58	0.68	0.52	0.58	0.50	0.57

Table B.2 (continued) The performance of SVM on retweet prediction task using different feed types

Embeddings Types	Precision		Recall		F1 score		Accuracy
	0	1	0	1	0	1	
Retweeted View (Tier 1)	0.10	0.44	0.17	0.72	0.11	0.50	0.41
Retweeted View (Tier 3)	0.36	0.03	0.80	0.20	0.49	0.05	0.38
Single-Feed (Tier 3)	0.04	0.02	0.78	0.10	0.54	0.04	0.43
Multi-Feed (Tier 3)	0.59	0.63	0.63	0.63	0.58	0.60	0.60

Table B.3 *Rank retrieval performance comparison between multi-feed and single-feed on the friendship recommendation task*

	Recall @K	Mean Reciprocal Rank
Multi-Feed LDA (Tier 1)	0.43	0.72
Single-Feed LDA (Tier 1)	0.38	0.47
TF-IDF LDA (Tier 1)	0.34	0.32
Multi-Feed LDA (Tier 2)	0.43	0.83
Single-Feed LDA (Tier 2)	0.39	0.47
TF-IDF LDA (Tier 2)	0.25	0.25
Multi-Feed LDA (Tier 3)	0.44	0.90
Single-Feed LDA (Tier 3)	0.40	0.85
TF-IDF LDA (Tier 3)	0.33	0.58

Table B.4 *Recall @K performance comparison between Canonical LDA and Twitter LDA on the three tiers of users*

	Recall @K
Twitter LDA (Tier 1)	0.18
Canonical LDA (Tier 1)	0.15
Twitter LDA (Tier 2)	0.24
Canonical LDA (Tier 2)	0.12
Twitter LDA (Tier 3)	0.20
Canonical LDA (Tier 3)	0.18

Table B.5 *Rank retrieval performance comparison between our MFWTE models vs. the multi-feed topic embeddings models on the friendship recommendation task*

	Precision @K	Recall @K	Mean Reciprocal Rank
MFWTE (Tier 1)	0.63	0.44	0.88
Non-weighted LDA (Tier 1)	0.54	0.38	0.72
Multi-view TF-IDF (Tier 1)	0.52	0.36	0.47
MFWTE (Tier 2)	0.44	0.67	0.86
Non-weighted LDA (Tier 2)	0.38	0.59	0.83
Multi-view TF-IDF (Tier 2)	0.36	0.49	0.68
MFWTE (Tier 3)	0.69	0.48	0.99
Non-weighted LDA (Tier 3)	0.62	0.44	0.95
Multi-view TF-IDF (Tier 3)	0.50	0.35	0.68

Table B.6 *Rank retrieval performance comparison between our MFWTE models vs. the multi-feed topic embeddings models on the retweet prediction task*

	Precision @K	Recall @K	Mean Reciprocal Rank
MFWTE (Tier 1)	0.59	0.23	0.50
RT-LDA (Tier 1)	0.55	0.21	0.20
Multi-feed LDA (Tier 1)	0.55	0.21	0.20
MFWTE (Tier 2)	0.47	020	0.70
RT-LDA (Tier 2)	0.40	0.16	0.35
Multi-feed LDA (Tier 2)	0.30	0.14	0.25
MFWTE (Tier 3)	0.54	0.34	0.50
RT-LDA (Tier 3)	0.61	0.37	0.80
Multi-feed LDA (Tier 3)	0.54	0.33	0.45

BIBLIOGRAPHY

Alonso-Mora, J., Breitenmoser, A., Rufli, M., Beardsley, P., & Siegwart, R. (2013). Optimal reciprocal collision avoidance for multiple non-holonomic robots. *Distributed autonomous robotic systems*, 203-216.

Andrew, G., Arora, R., Bilmes, J., & Livescu, K. (2013). Deep canonical correlation analysis. *International conference on machine learning*, (pp. 1247-1255).

Arbelaitz, O., Gurrutxaga, I., Muguerza, J., PéRez, J. M., & Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition, 46*(1), 243-256.

Armentano, M. G., Godoy, D., & Am, A. (2011). Recommending information sources to information seekers in Twitter. *Proceedings of the IJCAI 2011: International workshop on social web mining.*

Arora, R., & Livescu, K. (2014). Multi-view learning with supervision for transformed bottleneck features. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 2499-2503). IEEE.

Asteriou, D., & Hall, S. G. (2011). ARIMA models and the Box–Jenkins methodology. *Applied Econometrics 2, no. 2*, 265-286.

Backstrom, L., Huttenlocher, D., Kleinberg, J., & Lan, X. (2006). Group formation in large social networks: membership, growth, and evolution. *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 44-54). ACM.

Bakshy, E., Rosenn, I., Marlow, C., & Adamic, L. (2012). The role of social networks in information diffusion. *Proceedings of the 21st international conference on World Wide Web*, (pp. 519-528).

Bareiss, D., & van den Berg, J. (2015). Generalized reciprocal collision avoidance. *International Journal of Robotics Research, 34(12),* 1501-1514.

Bayındır, L. (2016). A review of swarm robotics tasks. *Neurocomputing, 172*, 292-321.

Begum, N., Ulanova, L., Wang, J., & Keogh, E. (2015). Accelerating dynamic time warping clustering with a novel admissible pruning strategy. *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 49-58). ACM.

Bellman, R. (1957). A Markovian decision process. *Journal of mathematics and mechanics*, 679-684.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. *Proceedings of the 26th annual international conference on machine learning* (pp. 41-48). ACM.

Ben-Hur, A., Horn, D., Siegelmann, H., & Vapnik, V. (2001). Support vector clustering. *Journal of Machine Learning Research 2*, 125-137.

Benton, A., Arora, R., & Dredze, M. (2016). Learning multiview embeddings of twitter users. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, (pp. 14-19).

Benton, A., Khayrallah, H., Gujral, B., Reisinger, D. A., Zhang, S., & Arora, R. (2019). Deep Generalized Canonical Correlation Analysis. *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)* (pp. 1-6). Florence, Italy: Association for Computational Linguistics. doi:10.18653/v1/W19-4301

Berndt, D. J., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. *KDD workshop, vol. 10, no. 16*, 359-370.

Bernstein, D. S., Givan, R., Immerman, N., & Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of operations research, 27(4)*, 819-840.

Bi, B., Tian, Y., Sismanis, Y., Balmin, A., & Cho, J. (2014). Scalable topic-specific influence analysis on microblogs. *Proceedings of the 7th ACM international conference on Web search and data mining*, (pp. 513-522).

Bird, S., Loper, E., & Klein, E. (2009). *Natural Language Processing with Python.* O'Reilly Media Inc.

Bishop, C. M. (1994). *Mixture density networks.*

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research 3, no. Jan*, 993-1022.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control.* John Wiley & Sons.

Boyd, D., Golder, S., & Lotan, G. (2010). Tweet, Tweet, Retweet: Conversational Aspects of Retweeting on Twitter. *Proceedings of the 2010 43rd Hawaii International Conference on System Sciences* (pp. 1–10). IEEE Computer Society. doi:10.1109/HICSS.2010.412

Bradtke, S. J., & Barto, A. G. (1996). Learning to predict by the method of temporal differences". *Machine Learning, 22*, 33–57. doi:10.1023/A:1018056104778

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M. (2013). Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence 7*, 1-41.

Breiman, L. (2001). Random forests. *Machine learning, 45(1),* 5-32.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees.* CRC press.

Bryson Jr, A. E., Denham, W. F., & Dreyfus, S. E. (1963). Optimal programming problems with inequality constraints. *AIAA journal, 1(11),* 2544-2550.

Carroll, J. D. (1968). Generalization of canonical correlation analysis to three or more sets of variables. *Proceedings of the 76th annual convention of the American Psychological Association (Vol. 3),* (pp. 227-228).

Chen, Y. F., Everett, M., Liu, M., & How, J. P. (2017). Socially aware motion planning with deep reinforcement learning. *arXiv:1703.08862.* arXiv.

Chen, Y. F., Liu, M., Everett, M., & How, J. P. (2017). Decentralized noncommunicating multiagent collision avoidance with deep reinforcement learning. *International Conference on Robotics and Automation*, (pp. 285–292).

Cireşan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural computation, 22(12),* 3207-3220.

Claes, D., Hennes, D., Tuyls, K., & Meeussen, W. (2012). Collision avoidance under bounded localization uncertainty. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1192-1198). IEEE.

Colbaugh, R., & Glass, K. (2012). Early warning analysis for social diffusion events. *Security Informatics, 1*(1), 18.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning 20, no. 3,* 273-297.

Ding, W., Li, S., Qian, H., & Chen, Y. (2018). Hierarchical reinforcement learning framework towards multi-agent navigation. *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)* (pp. 237-242). IEEE.

Dorigo, M., Birattari, M., & Brambilla, M. (2014). Swarm robotics. *Scholarpedia, 9(1)*, 1463.

Dunn, J. C. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics. 3 (3)*, 32-57.

Ferrara, E., & Yang, Z. (2015). Quantifying the effect of sentiment on information diffusion in social media. *PeerJ Computer Science, 1*, e26.

Fiorini, P., & Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research, 17(7)*, 760-772.

Firdaus, S. N., Ding, C., & Sadeghian, A. (2016). Retweet prediction considering user's difference as an author and retweeter. *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (pp. 852-859). IEEE.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1). New York: Springer series in statistics.

Garcia-Gavilanes, R., & Amatriain, X. (2010). Weighted content based methods for recommending connections in online social networks.

Gerkey, B., Vaughan, R. T., & Howard, A. (2003). The player/stage project: Tools for multi-robot and distributed sensor systems. *Proceedings of the 11th international conference on advanced robotics (Vol. 1)*, (pp. 317-323).

Godoy, J., Karamouzas, I., Guy, S., & Gini, M. (2016). Implicit coordination in crowded multi-agent navigation. *Thirtieth AAAI Conference on Artificial Intelligence.*

Goldberg, D., & Mataric, M. J. (2000). *Robust behavior-based control for distributed multi-robot collection tasks.* Los Angeles United States: University of Southern California.

Golder, S. A., Yardi, S., Marwick, A., & Boyd, D. (2009). A structural approach to contact recommendations in online social networks. *Workshop on search in social media.* SSM.

Goss, S., & Deneubourg, J. L. (1992). Harvesting by a group of robots. *Proceedings of the First European Conference on Artificial Life*, (pp. 195-204).

Granovetter, M. (1978). Threshold models of collective behavior. *American journal of sociology, 83*(6), 1420-1443.

Greene, D., & Cunningham, P. (2013). Producing a unified graph representation from multiple social network views. *Proceedings of the 5th annual ACM web science conference*, (pp. 118-121).

Gu, S., Lillicrap, T., Sutskever, I., & Levine, S. (2016). Continuous deep q-learning with model-based acceleration. *International Conference on Machine Learning*, (pp. 2829-2838).

Guille, A., Hacid, H., Favre, C., & Zighed, D. A. (2013). Information diffusion in online social networks: A survey. *ACM Sigmod Record 42, no. 2*, 17-28.

Guo, H., & Meng, Y. (2010). Distributed reinforcement learning for coordinate multi-robot foraging. *Journal of intelligent & robotic systems, 60(3-4)*, 531-551.

Gurini, D. F., Gasparetti, F., Micarelli, A., & Sansonetti, G. (2013). A Sentiment-Based Approach to Twitter User Recommendation. *RSWeb@ RecSys*, 1066.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290.* arXiv.

Hannon, J., McCarthy, K., & Smyth, B. (2011). Finding useful users on twitter: twittomender the followee recommender. *European Conference on Information Retrieval* (pp. 784-787). Berlin, Heidelberg: Springer.

Hao, D. N., & Lesnic, D. (2000). The Cauchy problem for Laplace's equation via the conjugate gradient method. *IMA Journal of Applied Mathematics 65, no. 2*, 199-217.

Haralabopoulos, G., Anagnostopoulos, I., & Zeadally, S. (2015). Lifespan and propagation of information in On-line Social Networks: A case study based on Reddit. *Journal of network and computer applications 56*, 88-100.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28, no. 1*, 100-108.

Hatua, A., Nguyen, T. T., & Sung, A. H. (2017). Information Diffusion on Twitter: Pattern Recognition and Prediction of Volume, Sentiment, and Influence. *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, (pp. 157-167).

Hebb, D. (1949). *The Organization of Behavior.* New York: Wiley.

Hecker, J. P., Letendre, K., Stolleis, K., Washington, D., & Moses, M. E. (2012). Formica ex machina: ant swarm foraging from physical to virtual and back again. *International Conference on Swarm Intelligence* (pp. 252-259). Berlin, Heidelberg: Springer.

Hennes, D., Claes, D., Meeussen, W., & Tuyls, K. (2012). Multi-robot collision avoidance with localization uncertainty. *AAMAS*, 147-154.

Ho, T. K. (1995). Random Decision Forests. *Proceedings of 3rd international conference on document analysis and recognition. 1*, pp. 278-282. IEEE.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation, 9*(8), 1735-1780.

Hoff, N. R., Sagoff, A., Wood, R. J., & Nagpal, R. (2010). Two foraging algorithms for robot swarms using only local communication. *2010 IEEE International Conference on Robotics and Biomimetics* (pp. 123-130). IEEE.

Hong, L., & Davison, B. D. (2010). Empirical study of topic modeling in twitter. *Proceedings of the first workshop on social media analytics*, (pp. 80-88).

Hüttenrauch, M., Adrian, S., & Neumann, G. (2019). Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research, 20(54*, 1-31.

Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika 32, no. 3*, 241-254.

Jurvetson, S., & Draper, T. (1997). Viral Marketing: Viral Marketing Phenomenon Explained. *DFJ Network News*.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence, 101(1-2),* 99-134.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research, 4*, 237–285. doi:doi:10.1613/jair.301

Kafeza, E., Kanavos, A., Makris, C., & Vikatos, P. (2014). Predicting information diffusion patterns in twitter. *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 79-89). Berlin, Heidelberg: Springer.

Kahn, G., Villaflor, A., Pong, V., Abbeel, P., & Levine, S. (2017). Uncertainty aware reinforcement learning for collision avoidance. *arXiv:1702.01182.* arXiv.

Kao, A. B., Miller, N., Torney, C., Hartnett, A., & Couzin, I. D. (2014). Collective learning and optimal consensus decisions in social animal groups. *PLoS computational biology 10 (8).*

Katz, E., & Lazarsfeld, P. (1970). *Personal Influence. The Part Played by People in the Flow of Mass Communications.* New Jersey: Transaction Publishers.

Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 137-146). ACM.

Kim, M., & Ramakrishna, R. (2005). New indices for cluster validity assessment. *Pattern Recognition Letters 26 (15),* 2353-2363.

Kraemer, L., & Banerjee, B. (2016). Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing, 190,* 82-94.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 1097-1105.

Krose, B. J., & Van Dam, J. W. (1992). Adaptive state space quantisation for reinforcement learning of collision-free navigation. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (Vol. 2)* (pp. 1327-1332).). IEEE.

Kwak, H., Lee, C., Park, H., & Moon, S. (2010). What is Twitter, a Social Network or a News Media? *Proceedings of the 19th International Conference on World Wide Web* (pp. 591-600). New York, NY, USA: Association for Computing Machinery.

Kywe, S. M., Lim, E. P., & Zhu, F. (2012). A survey of recommender systems in twitter. *International Conference on Social Informatics* (pp. 420-433). Berlin, Heidelberg: Springer.

Lazarsfeld, P., Berelson, B., & Gaudet, H. (1944). *The People's Choice: How the Voter Makes up his Mind in a Presidential Campaign.* New York: Columbia University Press.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436-444. doi:doi:10.1038/nature14539

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1(4)*, 541-551.

Letendre, K., & Moses, M. E. (2013). Synergy in ant foraging strategies: memory and communication alone and in combination. *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, (pp. 41-48).

Li, M., Wang, X., Gao, K., & Zhang, S. (2017). A survey on information diffusion in online social networks: Models and methods. *Information 8, no. 4*, 118.

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R news 2, no. 3*, 18-22.

Liu, L., Tang, J., Han, J., Jiang, M., & Yang, S. (2010). Mining topic-level influence in heterogeneous networks. *Proceedings of the 19th ACM international conference on Information and knowledge management*, (pp. 199-208).

Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H., & Pan, J. (2018). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 6252-6259). IEEE.

Long, P., Liu, W., & Pan, J. (2017). Deep-learned collision avoidance policy for distributed multiagent navigation. *Robotics and Automation Letters, 2*(2), 656-663.

Loper, E., & Bird, S. (2002). NLTK: the natural language toolkit. *arXiv preprint cs/0205028*.

Lu, Q., Moses, M., & Hecker, J. (2016). A scalable and adaptable multiple-place foraging algorithm for ant-inspired robot swarms. *Robotics: Science and Systems Conference (RSS 2016) workshop.*

MacQueen, J. (1967). Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability. 1*, pp. 281-297. University of California Press.

Macskassy, S. A., & Michelson, M. (2011). Why do people retweet? anti-homophily wins the day! *Fifth International AAAI Conference on Weblogs and Social Media.*

Makridakis, S., & Hibon, M. (1997). ARMA models and the Box–Jenkins methodology. *Journal of Forecasting 16, no. 3*, 147-163.

Marblestone, A. H., Wayne, G., & Kording, K. P. (2016). Toward an Integration of Deep Learning and Neuroscience. *Frontiers in computational neuroscience, 10,* 94. doi:10.3389/fncom.2016.00094

McCulloch, W., & Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics, 5*(4), 115-133. doi:doi:10.1007/BF02478259

Meilă, M. (2003). Comparing clusterings by the variation of information. *Learning theory and kernel machines* (pp. 173-187). Berlin, Heidelberg: Springer.

Miller, G. A. (1998). *WordNet: An electronic lexical database.* MIT press.

Mills, T. C. (1991). *Time series techniques for economists.* Cambridge University Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature, 518(7540),* 529-533.

Muller, U., Ben, J., Cosatto, E., Flepp, B., & Cun, Y. (2006). Offroad obstacle avoidance through end-to-end learning. *Advances in neural information processing systems*, 739–746.

Myers, S. A., Zhu, C., & Leskovec, J. (2010). Information diffusion and external influence in networks. *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining,* (pp. 33-41).

Naveed, N., Gottron, T., Kunegis, J., & Alhadi, A. C. (2011). Bad news travel fast: A content-based analysis of interestingness on twitter. *Proceedings of the 3rd international web science conference,* (pp. 1-7).

Nguyen, D. A., Tan, S., Ramanathan, R., & Yan, X. (2016). Analyzing information sharing strategies of users in online social networks. *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (pp. 247-254). IEEE.

Nguyen, T. T., & Banerjee, B. (2020). Reinforcement Learning as a Rehearsal for Foraging in Robot. *(in press)*.

Nguyen, T. T., Hatua, A., & Sung, A. H. (2019). Cumulative Training and Transfer Learning for Multi-Robots Collision-Free Navigation Problems. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)* (pp. 0305-0311). IEEE.

Nguyen, T. T., Hatua, A., Pothuraju, A., & Sung, A. H. (2018). Influence Modeling, Volume Prediction and Sentiment Analysis of Short Texts on Twitter. *Proceedings of 31st International Conference on Computer Applications in Industry and Engineering (CAINE 2018)*, (pp. 104-111). New Orleans, Louisiana, USA.

Nguyen, V. D., Varghese, B., & Barker, A. (2013). The royal birth of 2013: Analysing and visualising public sentiment in the uk using twitter. *2013 IEEE International Conference on Big Data* (pp. 46-54). IEEE.

Okubo, A. (1986). Dynamical aspects of animal grouping: swarms, schools, ocks, and herds. *Advances in biophysics 22*, 1-94.

Pak, A., & Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. *LREc, vol. 10, no. 2010*, 1320-1326.

Panait, L., & Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems, 11(3)*, 387-434.

Paudel, P., Hatua, A., Nguyen, T. T., & Sung, A. H. (2019). User Level Multi-feed Weighted Topic Embeddings for Studying Network Interaction in Twitter. *International Conference on Big Data* (pp. 80-94). Springer, Cham.

Pennacchiotti, M., & Gurumurthy, S. (2011). Investigating topic models for social media user recommendation. *Proceedings of the 20th international conference companion on World wide web*, (pp. 101-102).

Petitjean, F., Ketterlin, A., & Gançarski, P. (2011). A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition 44 (3)*, 678-693.

Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., & Birattari, M. (2012). ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence, 6(4)*, 271-295.

Pinto, J. C., & Chahed, T. (2014). Modeling multi-topic information diffusion in social networks using latent Dirichlet allocation and Hawkes processes. *2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems* (pp. 339-346). IEEE.

Prescott, T. J., & Mayhew, J. E. (1992). Obstacle avoidance through reinforcement learning. *Advances in neural information processing systems*, 523-530.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. *Machine learning*, 463-482.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning, 1(1)*, 81-106.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning.* Elsevier.

Ramage, D., Dumais, S., & Liebling, D. (2010). Characterizing microblogs with topic models. *Fourth international AAAI conference on weblogs and social media.*

Ramage, D., Hall, D., Nallapati, R., & Manning, C. D. (2009). Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: . Volume 1-Volume 1*, pp. 248-256. Association for Computational Linguistics.

Reagans, R., & McEvily, B. (2003). Network structure and knowledge transfer: The effects of cohesion and range. *Administrative science quarterly, 48*(1), 240-267.

Roesslein, J. (2009). *Tweepy*. Retrieved 04 20, 2020, from http://docs.tweepy.org/en/latest

Rogers, E. M. (2010). *Diffusion of innovations.* Simon and Schuster.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review, 65*(6), 386.

Rossi, F., Bandyopadhyay, S., Wolf, M., & Pavone, M. (2018). Review of multi-agent algorithms for collective behavior: a structural taxonomy. *IFAC-PapersOnLine, 51(12)*, 112-117.

Rousseeuw, P., & Kaufman, L. (1987). Clustering by means of medoids. *Statistical Data Analysis Based on the L1–Norm and Related Methods, edited by Y. Dodge, North-Holland*, 405–416.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature, 323(6088)*, 533-536.

Russell, S., & Norvig, P. (2009). *Artificial intelligence: a modern approach.* Prentice Hall.

Şahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. *International workshop on swarm robotics* (pp. 10-20). Berlin, Heidelberg: Springer.

Saito, K., Nakano, R., & Kimura, M. (2008). Prediction of information diffusion probabilities for independent cascade model. *International conference on knowledge-based and intelligent information and engineering systems* (pp. 67-75). Berlin, Heidelberg: Springer.

Saitta, S., Raphael, B., & Smith, I. F. (2007). A bounded index for cluster validity. *International workshop on machine learning and data mining in pattern recognition* (pp. 174-187). Berlin, Heidelberg: Springer.

Sardá-Espinosa, A. (2017). Comparing time-series clustering algorithms in r using the dtwclust package. *R package vignette 12*, 41.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning*, (pp. 1889-1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shakarian, P., Bhatnagar, A., Aleali, A., Shaabani, E., & Guo, R. (2015). The independent cascade and linear threshold models. *Diffusion in Social Networks*, 35-48.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., & Dieleman, S. (2016). Mastering the game of Go with deep neural networks and tree search. *nature, 529(7587)*, 484.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., . . . Chen, Y. (2017). Mastering the game of go without human knowledge. *Nature, 550*(7676), 354-359.

Simonin, O., Charpillet, F., & Thierry, E. (2014). Revisiting wavefront construction with collective agents: an approach to foraging. *Swarm Intelligence, 8(2)*, 113-138.

Snape, J., Berg, J. V., Guy, S., & Manocha, D. (2011). The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics, 27*(4), 696–706.

Snape, J., Van Den Berg, J., Guy, S. J., & Manocha, D. (2010). Smooth and collision-free navigation for multiple robots under differential-drive constraints. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 4584-4589). IEEE.

Sun, J., & Tang, J. (2013). Models and algorithms for social influence analysis. *Proceedings of the sixth ACM international conference on Web search and data mining*, (pp. 775-776).

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning (PhD thesis).* Amherst, MA: University of Massachusetts.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Tai, L., Paolo, G., & Liu, M. (2017). Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. *International Conference on Intelligent Robots and Systems.*

Tan, Y., & Zheng, Z. Y. (2013). Research advance in swarm robotics. *Defence Technology, 9(1),* 18-39.

Tang, D., Wei, F., Yang, N., Zhou, M., Liu, T., & Qin, B. (2014). Learning Sentiment-Specific Word Embedding for Twitter Sentiment Classification. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1555-1565). Baltimore, Maryland: Association for Computational Linguistics. doi:http://dx.doi.org/10.3115/v1/P14-1146

Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., & Kappas, A. (2010). Sentiment strength detection in short informal text. *Journal of the American society for information science and technology 61, no. 12*, 2544-2558.

Ulam, P., & Balch, T. (2003). Niche selection for foraging tasks in multi-robot teams using reinforcement learning. *2nd International Workshop on the Mathematics and Algorithms of Social Insects.*

Valetini, G., Ferrante, E., & Dorigo, M. (2017). The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI 9*.

Van Den Berg, J., Guy, S. J., Lin, M., & Manocha, D. (2011). Reciprocal n-body collision avoidance. *Robotics research*, 3-19.

Van den Berg, J., Lin, M., & Manocha, D. (2008). Reciprocal velocity obstacles for real-time multi-agent navigation. *2008 IEEE International Conference on Robotics and Automation* (pp. 1928-1935). IEEE.

Van Essche, S., Ferrante, E., Turgut, A., Lon, R., Holvoet, T., & Wenseleers, T. (2015). *Environmental factors promoting the evolution of recruitment strategies in swarms of foraging robots.*

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Thirtieth AAAI conference on artificial intelligence.*

Van Otterlo, M., & Wiering, M. (2012). Reinforcement learning and markov decision processes. Reinforcement Learning. *Adaptation, Learning, and Optimization, 12*, 3-42. doi:10.1007/978-3-642-27645-3_1

Wallace, M. (2007). *Jawbone Java WordNet API*. Retrieved from http://mfwallace.googlepages.com/jawbone

Wang, G., Hu, Q., & Yu, P. S. (2012). Influence and similarity on heterogeneous networks. *Proceedings of the 21st ACM international conference on Information and knowledge management*, (pp. 1462-1466).

Wang, N., Varghese, B., & Donnelly, P. D. (2016). A machine learning analysis of Twitter sentiment to the Sandy Hook shootings. *2016 IEEE 12th International Conference on e-Science (e-Science)* (pp. 303-312). IEEE.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning, 8(3-4)*, 279-292.

Weng, J., Lim, E.-P., Jiang, J., & He, Q. (2010). Twitterrank: finding topic-sensitive influential twitterers. *Proceedings of the third ACM international conference on Web search and data mining*, (pp. 261-270).

Werbos, P. (1974). *Beyond regression: new tools for prediction and analysis in the behavioral sciences.* Harvard University.

Wilkins, N. (2019). *Artificial Intelligence What You Need to Know About Machine Learning, Robotics, Deep Learning, Recommender Systems, Internet of Things, Neural Networks, Reinforcement Learning, and Our Future.*

Wu, J., Xu, X., Wang, J., & He, H. G. (2011). Recent advances of reinforcement learning in multi-robot systems: a survey. *Control and Decision, 26(11),* 1601-1610.

Xu, Z., & Yang, Q. (2012). Analyzing user retweet behavior on twitter. *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (pp. 46-50). IEEE.

Yang, J., & Leskovec, J. (2010). Modeling information diffusion in implicit networks. *2010 IEEE International Conference on Data Mining* (pp. 599-608). IEEE.

Yogeswaran, M., & Ponnambalam, S. G. (2012). Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task. *Opsearch, 49(3),* 223-236.

Zhang, J., Springenberg, J., Boedecker, J., & Burgard, W. (2016). Deep reinforcement learning with successor features for navigation across similar environments. *arXiv preprint arXiv:1612.05533*.

Zhao, W. X., Jiang, J., Weng, J., He, J., Lim, E.-P., Yan, H., & Li, X. (2011). Comparing twitter and traditional media using topic models. *European conference on information retrieval* (pp. 338-349). Berlin, Heidelberg: Springer.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J., Gupta, A., Fei-Fei, L., & Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning.

2017 IEEE international conference on robotics and automation (ICRA) (p. 3357). IEEE.

Zou, W. Y., Socher, R., Cer, D., & Manning, C. D. (2013). Bilingual Word Embeddings for Phrase-Based Machine Translation. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 1393-1398). Seattle, Washington, USA: Association for Computational Linguistics.

Zuo, Y., Zhao, J., & Xu, K. (2016). Word network topic model: a simple but general solution for short and imbalanced texts. *Knowledge and Information Systems 48, no. 2*, 379-398.