2023

Imitation Learning for Swarm Control using Variational Inference

Hafeez Olafisayo Jimoh
hoj00001@mix.wvu.edu

Imitation Learning for Swarm Control using Variational Inference

Hafeez Olafisayo Jimoh

A thesis submitted
to the Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the Requirements for the degree of

Masters of Science in
Mechanical Engineering

Dr Dimas Abreu Archanjo Dutra, Chair
Dr Guilherme A. S. Pereira
Dr Jason Gross

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2023

Keywords: Imitation Learning, Variational Inference, Graph Neural Network, Multi-robot systems

ABSTRACT

Imitation Learning for Swarm Control using Variational Inference

Hafeez Jimoh

Swarms are groups of robots that can coordinate, cooperate, and communicate to achieve tasks that may be impossible for a single robot. These systems exhibit complex dynamical behavior, similar to those observed in physics, neuroscience, finance, biology, social and communication networks, etc. For instance, in Biology, schools of fish, swarm of bacteria, colony of termites exhibit flocking behavior to achieve simple and complex tasks. Modeling the dynamics of flocking in animals is challenging as we usually do not have full knowledge of the dynamics of the system and how individual agent interact. The environment of swarms is also very noisy and chaotic. We usually only can observe the individual trajectories of the agents.

This work presents a technique to learn how to discover and understand the underlying governing dynamics of these systems and how they interact from observation data alone using variational inference in an unsupervised manner. This is done by modeling the observed system dynamics as graphs and reconstructing the dynamics using variational autoencoders through multiple message passing operations in the encoder and decoder. By achieving this, we can apply our understanding of the complex behavior of swarm of animals to robotic systems to imitate flocking behavior of animals and perform decentralized control of robotic swarms. The approach relies on data-driven model discovery to learn local decentralized controllers that mimic the motion constraints and policies of animal flocks. To verify and validate this technique, experiments were done on observations from schools of fish and synthetic data from boids model.

Acknowledgement

In the name of Allah, the Most Beneficent, the Most Merciful;

I give thanks to Almighty Allah, Al-Aleem- The All-Knower, Al-Khabeer- The All-Aware who has dominion over all things and who has provided the knowledge to undergo this project. Indeed, He has created all in the most beautiful and perfect way without any mathematical equation, literature review or supervisor. This topic stems from understanding and taking inspiration from flocking motion behavior in biology and implementing or mimicking such structures in multi-robots system using decentralized control. In converse to what some scientists say that believing in existence of Allah and science cannot co-exist, my research work proved to me that progress in science comes from acknowledging the perfection of Allah creation and taking motivation from such creation. It is no wonder that many recent advancements in robotics are biologically inspired. Allah says in Q67 vs 19 : Have they not seen the birds above them, spreading and folding their wings? None holds them up except the Most Compassionate. Indeed, He is All-Seeing of everything.

First, I would like to express my gratitude towards my advisor Dr Dimas Dutra for his mentorship and guidance throughout the course of this project. I would like to appreciate Dr Pereira for allowing me to join his group and Dr Jason Gross for agreeing to be a member of my committee. I thank all my friends and colleagues, in particular Uthman Olawoye, Juan Pabon, Ayooluwa Akintola, Ibrahim Oladepo and Mustopha Animashaun and family, and the entire MAE faculty in WVU who I have interacted with to provide an all round experience for me during the course of my program. I also acknowledge the computational resources provided by the WVU Research Computing Thorny Flat HPC cluster, which is funded in part by NSF OAC-1726534.

I thank my amazing parents and wonderful siblings that have been available to provide emotional and psychological support throughout the course of this 2 year journey coming to an end. Special shout-out goes to my beautiful wife, Safiyyah Idowu, and my daughter, Maymunah, that I was blessed with towards the start of this program. Both of them have given me reason everyday to always push further and better.

Lastly, I'd like to give credit to all the unnamed colleagues, my brothers at Islamic Centre of Morgantown(ICM) who have supported me, friends and family who often endured

my absence, and at other times provided me with the much-needed moral support and necessary distraction.

List of Figures

Table of Contents

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, multi-robot systems have gained significant attention in various fields [1], [2] including manufacturing, transportation, exploration, and surveillance. These systems consist of multiple autonomous robots that collaborate to achieve complex tasks more efficiently and effectively than individual robots. Decentralized control, where each robot makes decisions independently based on local information, is a promising approach to achieve robust and scalable multirobotic systems.

However, designing decentralized control strategies for multirobotic systems is a challenging problem due to the inherent complexity and uncertainty in the environment. Traditional approaches often rely on explicit modeling and explicit communication among robots, which can be computationally expensive, prone to errors, and difficult to scale up. Therefore, there is a growing interest[3] in developing learning-based methods that can enable robots to acquire control policies directly from observations and adapt to dynamic environments.

Animals like fish and social insects are known to coordinate their actions to accomplish tasks that surpass the capabilities of a single individual. For example, termites build large and complex mounds, army ants organize impressive foraging raids, ants can collectively carry large prey, and bees regulate the temperature of a hive. These animals rely on emergent behavior for their daily survival and exhibit fascinating complex behavior that is efficient, flexible, and robust [4]. The question is can we learn the dynamics of biological swarms and

apply such dynamics for decentralized control of multi-robots system.

How the collective emergent behavior observed in animals arise from local interactions between individuals is not fully understood. Biological swarms have limited and basic local information about their environment, as well as limited communication capabilities. The dynamics of animal motion and interactions between individuals are not explicitly known either. Despite the absence of a centralized controller, these animals exhibit collective behavior that includes flocking, formation, task allocation, collective transport, tolerance to the loss of a group member etc. Moreover, insect colonies, for example, demonstrate flexibility and robustness, which are highly desirable features in artificial systems. These natural examples, with their evident advantages, serve as a significant source of motivation for robotic swarms.

Many early works in swarm robotics relied on some form of centralized control and employed a bottom-up approach to designing controllers. In centralized control, a controller computes control actions based on knowledge of the global state obtained from all agents. There is a unit or agent responsible for planning the states of all other agents and providing trajectories for the entire system, which are then communicated to individual agents. However, this approach presents challenges such as scalability and a single point of failure. Swarm robots are intended to scale to an arbitrary number of robots, as a result the failure of a single member or the controller itself may result in the failure of the entire group. Additionally, global communication capabilities may be impossible or unfeasible under certain conditions, posing a problem for the scalability of the collective system. Hence, decentralized control becomes a necessary and attractive alternative to address these challenges. Research in swarm robotics draws inspiration from decentralized control behaviors observed in animals and insects.

There are already attempts to engineer flocking behavior like Reynolds boids virtual agents [5] and Helbing model[6]. For instance, in [7] an example of how multi-robots imitated group of ants trying to move preys by inferring rules from the ants to cooperatively push a box. In a stick-pulling experiment from the ground, group of robots collaborated together by generating abstracted macroscopic models that capture the dynamics of the robotic swarm.

The idea behind this work is to emulate, in a robotic swarm, complex emergent behavior seen in schools of fish in multiple robots. We assume that the robots have limited sensing and communication capabilities. By using a data driven bayesian learning technique

known as variational inference, we can capture the latent variables for understanding the interactions and dynamics of schools of fish. Also, since the model parameters of the schools of fish are not known, Variational Inference (VI), allows to approximate the posterior distribution which is our guess of the observations over the model parameters. VI relies on bayes' theorem that expresses the relationship between updated knowledge (the posterior), prior knowledge and the knowledge coming from observation (the likelihood). The observations (animal motion) are abstracted as graphs which allow to capture the inter-relationships between each agent in the system. By designing an encoder and decoder model and using VI to generate our objective function, we are able to come up with predictions of future state of animals given current state as well as determine the interactions between them. In addition, the learned model allows us to apply it as controller for implementing motion policies in robotic swarms.

1.2 Problem Statement

Schools of fish are challenging to learn and understand because they exhibit non-linear dynamics, chaotic and noisy complex collective behavior that arises from individual interactions. However, the swarm behavior of schools of fish are desired in robotics because it is scalable for large number of robots, robust, flexible, reactive and tolerant to failure of a member. The fish dynamics is not well understood, highly non-linear, noisy and chaotic. Imitation learning with variational inference can allow us to learn dynamics of swarming behavior in biological swarms. Thus, there is need for a system that can properly learn how to model the dynamics of the schools of fish and come up with predictions of their next state given a current state as well as the latent interactions between them.

This thesis aims to address this objective by proposing a graph variational autoencoder-based approach to imitation learning of schools of fish. The proposed approach will incorporate the graph structure of the fish interactions to learn a low-dimensional latent representation of the collective behavior, which can then be used for imitation learning. The approach will be evaluated on synthetic data from boids model and real-world data from schools of fish to demonstrate its effectiveness and generalizability. The outcomes of this research will contribute to our understanding of collective behavior and provide a foundation

for the development of more effective imitation learning methods for schools of fish.

Overall, this thesis aims to contribute to the field of swarm robotics by proposing a method for modeling the dynamics of schools of fish that is scalable, effective and efficient. By demonstrating the feasibility of this approach, the learned dynamics can then be used to generate motion policies to be used as control actions to our robotics system and the interaction graphs can help us determine how robot can communicate with each other effectively while still exhibiting rich sets of motion to achieve their tasks.

1.3 Objectives

The core objective of this work is to develop a methodology for learning dynamics of swarm motion. In order to achieve this, the following specific objectives would be met:

- Develop a variational inference learning technique to discover the dynamics of biological swarm motion
- Develop a variational inference learning technique to discover the dynamics of observation data from boids model
- Validate emergent behavior in swarms for predicting future trajectories and free run simulation using discovered dynamics from animal motion.

Chapter 2

Background and Literature Review

2.1 Swarms Robotics and Multi-Robot Systems

According to [8], swarms intelligence is the discipline that that deals with natural and artificial systems composed of many individuals that coordinate using decentralized control and self-organization. In particular, the swarm robotics discipline focuses on the collective behaviors that result from the local interactions of the individuals with each other and with their environment. Swarm robotics is the study of how large number of relatively simple physically embodied agents can be designed such that a desired collective behavior emerges from the local interactions among agents and between the agents and the environment. Swarms robotics are often inspired by natural systems -like schools and flocks of birds,ants - in which large numbers of simple agents are modeled after swarms of insect or other animals to perform complex collective behavior. Like in biological swarms, swarm robotics often involve use of decentralized control mechanisms where each robots operate independently and communicate with its neighbors to coordinate their behavior. Through the use of relatively simple rules and local interactions, the goal of swarms robotics is to develop robust, scalable, and flexible collective behaviors for the coordination or collaboration or cooperation of large number of robots [9]. Swarms robotics is closely related to Multi robot systems because both involves the use of multiple robots. The line between swarms robotics and multi-

robotic system is blur and there is a significant overlap between the two fields because both use multiple robots to achieve a task. Multi-robotic systems for instance borrow ideas and techniques from swarms intelligence. Multi robotic system (MRS) is composed of collection of 2 or more robots tasked with capability of doing tasks a single robot is not capable of doing. MRS can have centralized control or decentralized control mechanisms and can involve homogeneous or heterogeneous group of robots with same or different capabilities.

By applying time varying signals and networks to aggregation graph neural network, imitation learning of global information of centralized controllers were used to learn decentralized controllers that require local information and communication [10]

2.2 Learning dynamics and trajectories using Graph Neural Networks

Graphs as shown in Fig. 2.1 consist of nodes and edges, where nodes represent entities (e.g., atoms in a molecule, words in a sentence, a robot in a swarm etc), and edges represent relationships or connections between nodes. The original graph neural network (GNN) model for learning graph structured data was introduced in 2009. The work [11] presented a supervised neural network model designed as an extension of recursive neural networks and random walk models that can process cyclic, directed and undirected graphs. Message Passing Neural Network is a type of neural network architecture that is designed for learning on graph-structured data. The fundamental building block that other recent graph neural network models build on is the message neural network (MPNN). MPNN has appeared in a couple of works in interaction networks in learning molecular structures, predicting links etc. [12]–[17]. The MPNN framework is built upon the concept of message passing, which involves passing information or messages between nodes in a graph. It operates in a series of iterative steps, where each step updates the representations of nodes based on the information from neighboring nodes.

The gated graph sequence neural network (GGNN) [18] is an extension of GNN that combines the strengths of recurrent neural networks and graph based models. The key idea behind GGNN is the use of gated recurrent units (GRUs) to update the hidden states of

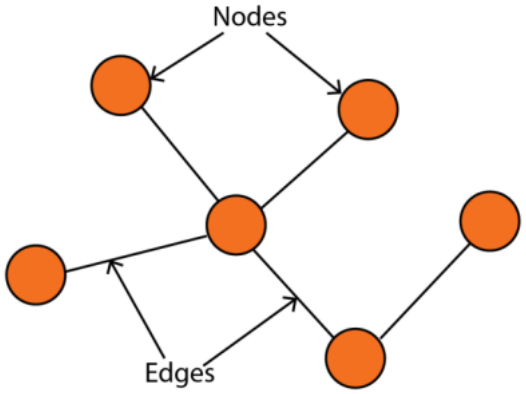


Figure 2.1: Pictorial Representation of Graphs

nodes. GRUs are a variant of RNNs (recurrent Neural Network) that incorporate gating mechanisms, which selectively control the flow of information and facilitate capturing long-range dependencies. The GGNN employs these gating mechanisms to control the flow of information during the iterative updates, enabling the network to focus on relevant information from the neighborhood of each node. It outputs sequences like paths on a graph and not just single outputs useful for graph-level classification. At each iteration, the GGNN takes as input the current hidden states of nodes and the graph's adjacency matrix. It computes an update gate and a reset gate for each node based on its current hidden state and the hidden states of its neighbors. These gates determine how much information from the previous iteration should be preserved or discarded. The hidden states of nodes are then updated by blending the previous hidden states with the computed update gate and reset gate. This allows the network to learn how much of the new information should replace the old information and how much of the previous hidden state should be retained.

Many of the aforementioned previous work are designed for problems with static graph structures whose signals or relations are not dynamic. We have only considered graphs where edges and features do not change. However, in the real world, there are many applications where this is not the case. For instance, in social networks, people follow and unfollow other users, posts go viral, and profiles evolve over time. In multi-robot system, a robot may be attacked by an adversarial and the robot states evolve over time. This dynamic property cannot be represented using the GNN architectures previously described in earlier paragraph. Instead, we must embed a new temporal dimension to transform static graphs

into dynamic ones. These dynamic networks will then be used as inputs for a new family of GNNs: Temporal Graph Neural Networks (T-GNNs), also called Spatio-Temporal GNNs.

There are two categories of graphs with temporal signals generally: static graphs with temporal signals where the underlying graph does not change, but features and labels evolve over time and dynamic graphs with temporal signals where the topology of the graph (the presence of nodes and edges), features, and labels change over time. For example, it can represent a network of cities within a country for traffic forecasting: features change over time, but the connections stay the same. In the second option, nodes and connections are dynamic. It is applicable in social networks for instance

In recent time, there have been a number of work that deals with GNNs for spatio-temporal signals on dynamic and static graphs. In Message Passing Neural Network with Long Short Term Memory (MPNNLSTM) [19], two layers of LSTM were combined with MPNN for COVID-19 epidemiological prediction. Nodes corresponds to different countries, and edge weights denote represented total number of people that moved from one region to another and GNN were employed to predict future cases. Diffusion Convolutional Recurrent Neural Network (DCRNN) [20] introduced a graph based deep learning framework for traffic forecasting and other spatiotemporal forecasting tasks. The diffusion convolutional layer operates by iteratively aggregating information from neighboring nodes and updating the node representations. It follows a message-passing scheme, where each node aggregates information from its neighboring nodes, combines it with its own representation, and produces an updated representation. DCRNN is an autoencoder based model that combines RNN with diffusion convolution in the encoder and decoder layer to come up with predictions. In Dynamic Graph Autoencoder (DyGrAE) [21], a GGNN was incorporated with LSTM using an encoder-decoder framework. The process involves a GGNN that allows to capture graph topology over time and recurrent LSTM layers propagate the temporal informatioon across the node at at each timestep. The encoder learns a latent representation and the decoder auto-regressively reconstructs the dynamic graph structure using the latent representation. DyGrAE was evaluated on dataset of animal behaviour and brain networks and result shows the model achieves significant better performance compared to the state-of-the-art models that learn static graph representation. There are a number of other works in literature [21]–[26] that also deals with spatio-temporal forecasting of graph datasets which one can

implement the techniques from them and apply to learning dynamical model of biological swarms.

2.3 Learning Swarms behavior

There have been recent trend in the past few years to understand complex animal behaviours and apply the learnt policies to robotics system using machine learning techniques [27]–[29]. Deep learning has enables tremendous progress in the area of pattern recognition of very complex and high dimensional data. In some other works, multi-agent systems like social networks, protein synthesis and molecule generation, there have been techniques to understand how local interactions give rise to emergent complex behaviours [30]. This review discusses specific related work that have applied reinforcement learning, graph neural networks and sparse regression technique to learn and imitate multi-agent systems or interacting systems dynamics.

By using Inverse Reinforcement Learning, the work in [27] developed a technique to predict the most likely route that an animal would have traveled by learning a reward function from animal trajectories to determine most preferable environmental features affecting locomotion. In this study, the agents and action sequences correspond to animals and their trajectories respectively, and the IRL algorithm provides a a trajectories shearwaters prefer to follow. Using an actor critic approach for deep reinforcement learning [31], the control of cooperative agents with limited sensing capabilities was investigated. The global position of the agents is available to the critic but the actor only base decision on locally sensed information. This work experiment was done in a simulated swarm environment using the Kilobot robot platform. An LSTM network was trained to learn temporal dependencies given a subset of animal trajectories [29]. The idea is to use the trained network as a motion planning to be implemented on robots such that they can operate autonomously without knowing their absolute position in a global frame. A neural network model based on genetic algorithm was used in [28] to reproduce postural information and classify behavioral pattern of zebra fish. There was no attempt to learn the interactions between the zebrafish. Path-finding via Reinforcement and Imitation Multi-Agent Learning (PRIMAL) [32] is a method that combines both reinforcement and imitation learning to teach multi-agent systems fully

decentralized policies. In a randomized partially observable world, 1024 agents reactively planned their paths. By making a Graph Convolutional Network (GCN) exploits the graph structure of the agents in their RL control policy, the work in [32] was extended in [33]. The GCN does dimensionality reduction by learning message passing functions that aggregates and update information among the agents. Reinforcement based learning techniques is however not usually suitable for learning emergent behaviors of swarms because of challenge of defining reward functions that rely on well-defined tasks [34].

Similar to [33], [35] was one of the foremost work that applied graph convolutional network in form of variational autoencoders to interacting systems (particles connected by springs, charged particles and phase-coupled oscillators (Kuramoto model) for learning interactions between different nodes while simultaneously learning the dynamical model of the interacting. Neural relational inference for interacting systems (NRI) [35] was one of the very first work that inferred interpretable interactions between entities explicitly. Other works like [36] and [37] only inferred interactions implicitly and the system dynamics could not be inferred. NRI suffers from assuming that the graph network is static over time. Dynamic neural relational inference (dNRI) [38] built upon this work and introduced a technique of generating graph embeddings or latent variables for every point in time. This is more practical for many interacting systems like animal swarms. The dNRI approach was demonstrated on synthetic particle, human motion capture, basketball player, and traffic trajectory datasets with significant improvement over baseline LSTM based methods and NRI previously mentioned. SwarmNet [39] is a variant of the decoder part of [35]. It however to efficiently discover the swarm dynamics from positions and velocities of a set of agents.

In Sparse Identification of Nonlinear Dynamics (SINDy) [40], data driven approach with machine learning technique was used to determine the governing equations from noisy measurement data with a strict assumption of that the governing equations are sparse in a high-dimensional nonlinear function space. SINDy was developed on the foundations of sparse regression via lasso regularization [41], [42] and compressed sensing [43]–[45] and can incorporate partial knowledge of the physics, such as symmetries, constraints, and conservation law. An extension to Sindy is the implicit sindy [46] which is more suited for more complex biological networks and solves the SINDy difficulty in discovering implicit dynamics and rational functions. A more robust variant of SINDy and Implicit SINDy is the SINDy-

PI(Parallel and implicit) [47]. SINDy-PI includes a constrained optimization algorithm for each candidate function selection and a rich approach to model selection. SINDy-PI could learn implicit ordinary and partial differential equations and conservation laws from limited and noisy data. This approach was demonstrated on 2-link pendulum, actuated pendulum on cart, the Belousov–Zhabotinsky PDE, and the identification of conserved quantities.

While approaches like [35], [38], [39] give good approximations of original dynamic and interpretable interactive graphs, the predicted dynamics is not interpretable. Techniques discussed in [40], [46], [47] allows to discover interpretable implicit and explicit governing dynamics equations, it may not perform well like machine learning based approximators techniques using graph convolutional network. SINDy Autoencoders [48] uses a combination of the autoencoder and SINDY approach to generate rich low dimensional interpretable dynamical model from high dimensional observation data alone. Using agent physics neural network termed knowledge-based neural ordinary differential equations (KNODE), decentralized controllers in single-robots were used to mimic flocking behavior in swarm.

This thesis relies heavily on the techniques used in [35] and [38]. However, there is no currently no work done yet, based on the available knowledge to the author, on applying these techniques to observation data from schools of fish with very complex yet emergent behavior. A closely related work is [39], but the experiment in it was only done on artificially simulated swarms that are unable to perfectly model the intricacies in biological swarms. The fish dataset used is available in [49].

2.4 Variational Inference as Deep State Space Models

Deep state space models are a class of machine learning models that combine elements of deep learning and state space models. State space models (SSMs) are probabilistic models used to describe the evolution of a latent (unobserved) state over time, along with the observed measurements associated with the state. Deep learning models, on the other hand, are neural network architectures capable of learning complex patterns and representations from data.

Deep state space models (SSMs) aim to leverage the strengths of both approaches by incorporating deep neural networks into the state space modeling framework. In these

models, the latent state is often represented by a recurrent neural network (RNN), such as a long short-term memory (LSTM) or a Gated Recurrent Unit (GRU). The RNN captures the temporal dependencies and dynamics of the latent state over time. The observations in deep state space models can be either discrete or continuous, and they are typically modeled as a probabilistic distribution conditioned on the current state. The parameters of the observation model are learned from the data using techniques such as maximum likelihood estimation or variational inference.

Deep State Space Models for Nonlinear System Identification [50] treated the problem as nonlinear system identification and introduced parameter learning for deep SSMs by combining RNNs with variational autoencoders. The VAE are used to approximate the output distributions of the dynamics from the RNN output. The approach in this work is very similar to the technique presented in [50]. This work builds on it and apply the technique to graph neural network with a graph data structure as input. In the next section of this chapter, backgrounds to what graphs are would be introduced briefly since this is a fundamental building block used int he methodology of this project. A number of literature references also rely on graph theory.

2.5 Introduction to Graphs

2.5.1 Nodes, Edges and Weights

A graph is a mathematical abstraction of network structure that serves as a way of specifying the relationships and how information is shared among collection of different items in a network [51]. A graph typically consists of nodes with links defining the relations between nodes called edges. In mathematical parlance, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is defined by a finite set of nodes $\mathcal{V} = \{1, ...N\}$ and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ with weights $\mathcal{W}$. Edges are ordered pairs of labels (i, j). We interpret $(i, j) \in \mathcal{E}$ as i can be influenced by j. Weights $\mathbf{w}_{ij} \in \mathbb{R}$ are numbers associated to edges (i, j). Weights define the strength of the influence of j on i. Two nodes are said to be neighbors if they are connected by an edge. Graphs as shown in Fig. 2.2 are typically drawn with little circles representing the nodes and a line representing the connection between each pair of nodes. Graphs can appear in many domains

like computer networks, multi-robotic systems, protein molecules, biological swarms, traffic modeling, etc.

2.5.2 Adjacency and Degree Matrix representation

The adjacency matrix of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ is the sparse matrix A with nonzero entries $A_{ij} = w_{ij} \in \mathbb{R} \;\; \forall (i,j) \in \mathcal{E}$ An adjacency matrix is a matrix that represents the edges in a graph, where each cell indicates whether there is an edge between two nodes. The matrix is a square matrix of size $n \times n$, where is the number of nodes in the graph. A graph is unweighted if the adjacency matrix doesn't have weights. In that case, one can say that all weights are units; $w = 1 \forall (i,j)$. A value of 1 in the cell (i,j) indicates that there is an edge between node i and node j, while a value of 0 indicates that there is no edge. For an undirected graph, the matrix is symmetric, while for a directed graph, the matrix is not necessarily symmetric.

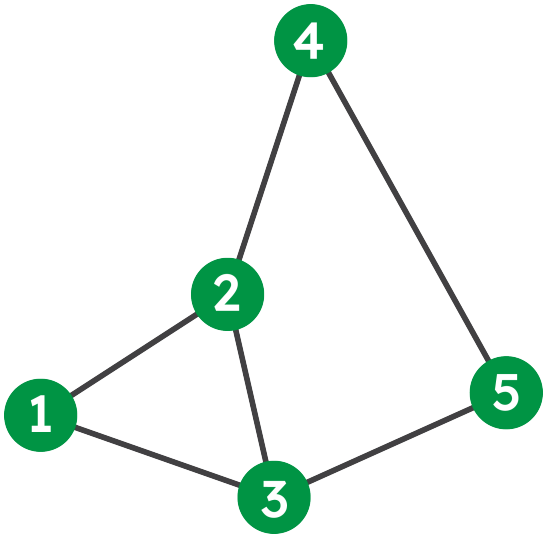


Figure 2.2: An undirected graph with 5 nodes

For example, the adjacency matrix of the graph in Fig. 2.2 can be written as:

$$A = \begin{bmatrix} 0 & w_{12} & w_{13} & 0 & 0 \\ w_{21} & 0 & w_{23} & w_{24} & 0 \\ w_{31} & w_{32} & 0 & 0 & w_{35} \\ 0 & w_{42} & 0 & 0 & w_{45} \\ 0 & 0 & w_{53} & w_{54} & 0 \end{bmatrix} \tag{2.1}$$

If the graph is unweighted, then;

$$A_{ij} = 1 \qquad (i,j) \in \mathcal{E} \tag{2.2}$$

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.3}$$

For an undirected graph $\mathcal{G}$, the degree of a given node or vertex, $d(v_i)$, is the cardinality of the neighborhood set $\mathcal{N}$, that is, it is equal to the number of vertices that are adjacent to node in $\mathcal{G}$. For example, for the graph in Fig. 2.2, the neighborhood sets are : $\mathcal{N}(v_1) = [v_2, v_2]$, $\mathcal{N}(v_2) = [v_1, v_3, v_4]$, $\mathcal{N}(v_3) = [v_1, v_2, v_5]$, $\mathcal{N}(v_4) = [v_2, v_5]$ and $\mathcal{N}(v_5) = [v_2, v_3]$ For each node, it is basically the sum of all adjacent neighbors connected to the node. For the graph in Fig. 2.2, the degree of the graph is: $d(v_1) = 2, d(v_2) = 3, d(v_3) = 3, d(v_4) = 2, d(v_5) = 2$, The degree matrix of $\mathcal{G}$ is the diagonal matrix, containing the vertex-degrees of $\mathcal{G}$ on the diagonal, that is:

$$D = \begin{bmatrix} d(v_1) & 0 & \dots & 0 \\ 0 & d(v_2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & d(v_n) \end{bmatrix} \tag{2.4}$$

where n is the number of nodes in the graph.

The adjacency matrix is a straightforward representation that can be easily visualized

as a 2D array. One of the key advantages of using an adjacency matrix is that checking whether two nodes are connected is a constant time operation. This makes it an efficient way to test the existence of an edge in the graph. Moreover, it is used to perform matrix operations, which are useful for certain graph algorithms, such as calculating the shortest path between two nodes.

2.5.3 Preliminaries: Machine Learning on Graphs

Machine learning models typically take grid-like arrays as inputs which could be structured 1D or 2D dimensional data or images and videos. Graph data structures are more more complex, have complex relationships and inter-dependency between objects and cannot be represented on euclidean space; graphs do not typically exist in 2D or 3D spaces like images or videos. Also, in traditional machine learning tasks with 2D or 3D data, the correlations between different variables are not modeled explicitly. This can lead to errors in situations where the predictions per-variable are uncertain. Whereas using a model which properly considers the correlations among variables can ameliorate this uncertainty which makes it challenging to model them with traditional machine learning techniques.

The task of machine learning is to learn some models from data and come up with some prediction. In supervised learning, the goal is to use the inputs to predict the values of the outputs. In unsupervised learning, the goal is to infer patterns or clusters of points in the data. In machine learning with graph data structures, the traditional categories of supervised and unsupervised categories are often blurred. The classical machine learning tasks shown in Figs. 2.3 and 2.4 are node classification, link or relation prediction, clustering and community detection.

Link prediction is a task that involves predicting missing or future links in a graph based on the existing links and node features. Node classification involves assigning labels or categories to nodes in a graph based on their features and the graph structure. Graph Neural Networks (GNNs) [52] have proven to be effective in node classification tasks by leveraging the connectivity patterns and node embeddings in the graph. Clustering using Graph Neural Networks (GNNs) involves grouping nodes in a graph into clusters or communities based on their structural connectivity and node features. GNNs have shown promise in capturing

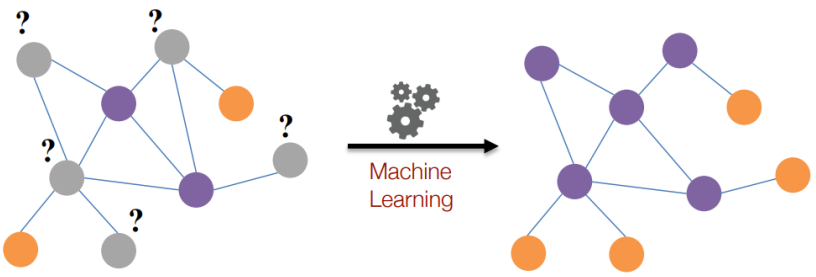


Figure 2.3: Node Classification

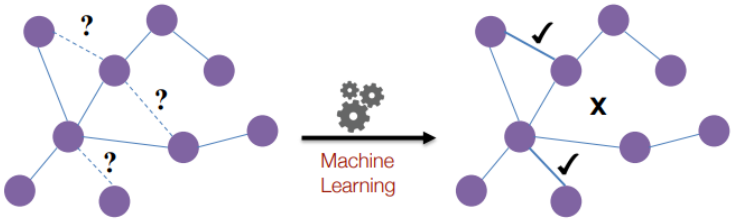


Figure 2.4: Link Prediction

the graph topology and node attributes to perform effective clustering tasks. There are several variations and extensions of GNNs including graph autoencoders, graph attention networks (GAT) [53], Graph Convolutional Networks (GCN) [54], and more. The choice of the specific GNN architecture for link prediction, node classification and clustering depends on the characteristics of the graph and the specific link prediction task at hand [52].

Chapter 3

Methodology

This chapter discusses the methodology developed in this work. It starts by giving a background on graph neural network as it relates to machine learning on graphs, and introduces variational autoencoders which is a neural network based technique of implementing variational inference. These are all techniques applied in this thesis.

3.1 Graph neural Networks

3.1.1 Message Passing

Message passing is the fundamental building block of a Graph Neural Network. Message passing Neural network was first proposed in [12]. It is analagous to a multi-layer perceptron in a neural network. Message passing involves summing the incoming message from neighbor and combining the aggregation with the node's previous embedding using a linear combination and finally applying a non-linearity like RELU. The intuition behind message passing is that every node shares information with its neighbors by aggregating information from its local neighbors and updating its embeddings. For example, after n iterations, every node contains information from its n-hop neighbor. In implementing this, the first step is to creates node feature vectors (for example position and velocity in x and y coordinate as used in this work) that represents the message the nodes wants to send to all its neighbors. In the second step, the messages are sent to the neighbors, so that a node receives one message per adjacent node. These messages are then aggregated by a permutation and

order invariant function (like sum or average) at each node. The resulting vector is passed through a neural network layer (which just means it is multiplied by some matrix $W^{(k)}_{neighb}$ and then a non-linear activation function is used on the result to get an updated feature state vector. The two steps for in message passing operation for an example graph is shown below. In Fig 3.1, we see how node 1 now contains information from its neighbor (node 2) and node 2 contains information from all its neighbors (node 1,2,3 and 4).This information is what is referred to as messages that are aggregated and used to update the state in each node. The message passing operation is described below and more details about it can be

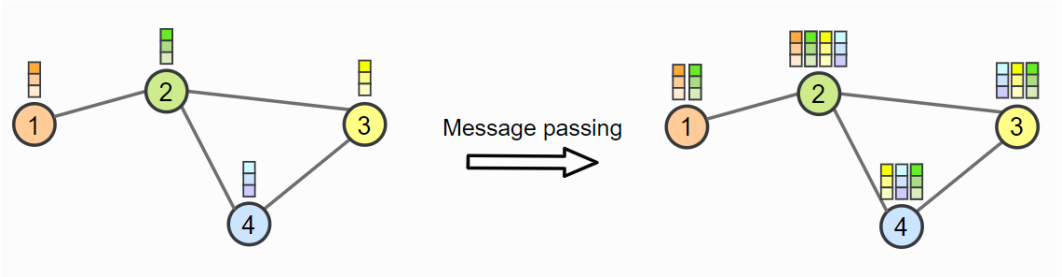


Figure 3.1: message passing operation: Image used with permission by the author [55]

found in [56]. The basic GNN message passing update can be expressed as:

$$h_u^t = \sigma\Big(W_{self}^t h_u^{t-1} + W_{neigh}^t \sum_{v \in \mathcal{N}(u)} h_v^{t-1} \Big) \tag{3.1}$$

where, the superscript t denotes the embedding functions at successive iterations of the message passing operation; h_u^t is an hidden embedding that corresponds to each node $u \in \mathcal{V}$. At each iteration t, messages are aggregated from all the u's graph neighbors $\mathcal{N}(u)$. h_u^{t-1} denotes the node u embedding on a graph in previous iteration and $h_v^{(t-1)}$ are the embeddings on the neighbors $v \in \mathcal{N}(u)$. At t $= 1$, $h_u^{(0)}$ represents the initial embeddings which is equivalent to the input features for all the nodes, i.e. $h_u^0 = \mathbf{x}_u, \forall\, u \in \mathcal{V}$. Thus, we can see that the GNN message passing neural network requires that we have the node features $\mathbf{x}_u, \forall\, u \in \mathcal{V}$ as input to the model. W_{self}^t and W_{neigh}^t are learnable neural network parameter matrices. $\sigma(\dots)$ also denotes a non-linear activation function (RELU or tanh). In more compact form, the message passing can be written as:

$$h_u^t = \text{UPDATE}(h_u^{t-1}, m_{\mathcal{N}(u)}^t) = \sigma(\, W_{self}^t h_u^{t-1} + W_{neigh}^t m_{\mathcal{N}(u)}^t) \tag{3.2}$$

$$m^t_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} h^{t-1}_v \tag{3.3}$$

where: $m^t_{\mathcal{N}(u)}$ is the "message" that is aggregated from u's graph at time t. neighborhood $\mathcal{N}(u)$

$$m^t_{\mathcal{N}(u)} = \text{AGGREGATE}(h^t_v, \forall\, v \in \mathcal{N}(u)) \tag{3.4}$$

Since W^t_{self} and W^t_{neigh} comes from a neural network, (3.1) can be written in terms of $f^{(t)}_{...}$ as:

$$h^t_u = f^t_u\left(h^{t-1}_u, f^t_{emb}\Big(\sum_{v \in \mathcal{N}(u)} h^{t-1}_v\Big)\right) \tag{3.5}$$

where f^t_u denotes the neural network for updating a node and f^t_{emb} denotes the neural network for aggregating embeddings from neighbors of a node. After t iteration the output of final embedding layer can be defined as:

$$\mathbf{z}_u = h^t_u, \forall\, u \in \mathcal{V} \tag{3.6}$$

By using a neural network to learn the weights transformation for the aggregation of the messages, graph neural networks can learn a non-linear function that captures these complex relationships and interactions. During training, the network is trying to determine how much modification through a non-linear transformation we need to apply on a current node feature and its neighbors. This enables the model to effectively leverage the rich structure of the graph and incorporate information from the node's local and global neighborhoods in a more expressive and flexible way.

Summarily, the basis of message passing neural algorithm is to find a non-linear function that updates node values. Just like any neural network, our goal is to find an algorithm to update these node values which is analogous to a layer in the graph neural network. And then one can of course keep on adding such layers. Just as node values get updated through message passing operation, edge values are also updated.

3.1.2 Graph Convolutional Network (GCN)

The message passing operation in previous section is the basic building block of the graph convolutional network. The GCN employs multiple layers of message passing operation as shown in Fig. 3.2 to propagate and update node representations by aggregating and transformation of neighbors information. In the GCN proposed in [54], the message passing operation simply had a normalization factor $c_{i,j}$ included in it

$$h_u^t = \sigma\Big(W_{self}^t h_{(u)}^{t-1} + W_{neighb}^t c_{i,j} \sum_{v \in \mathcal{N}(u)} h_v^{t-1} \Big) \tag{3.7}$$

where:

$$c_{i,j} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \tag{3.8}$$

In (3.8), $\tilde{A} = A + I_N$ is the adjacency matrix for the undirected graph with N nodes. The addition of the identity matrix corresponds to adding self loops to the graph. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ represents the degree matrix of the graph. The above definition of the normalization factor c_{ij} corresponds to that proposed in [54]. D_i represents the degree of node i and D_j represents the degree of node j. The logic behind GCNs is similar to that of conventional neural networks where the the output of one layer is fed to the next one as input. The number of layers in GCNs determine how far the messages from each node will be able to travel through the connections of the data structure. In a two-layer network, for example, the message passing operations happens twice and the signal will only do two hops from the source node and won't be affected by the info from outside of the subgraph. The specific of the problem determines how deep the number of GCN layers. At the same time, to prevent overfitting, the selection of depth of the GCN is carefully choosen and this can be regarded as an hyperparameter in the algorithm.

3.1.3 Gated Graph Network Network (GGNN)

The GGNN is based on the technique discussed in [15]. In a GGNN, each node in the graph is associated with a hidden state vector, and the connections between nodes are represented by edges. The GGNN updates the hidden state of each node by iteratively aggregating information from its neighboring nodes. The key idea in a GGNN is the use

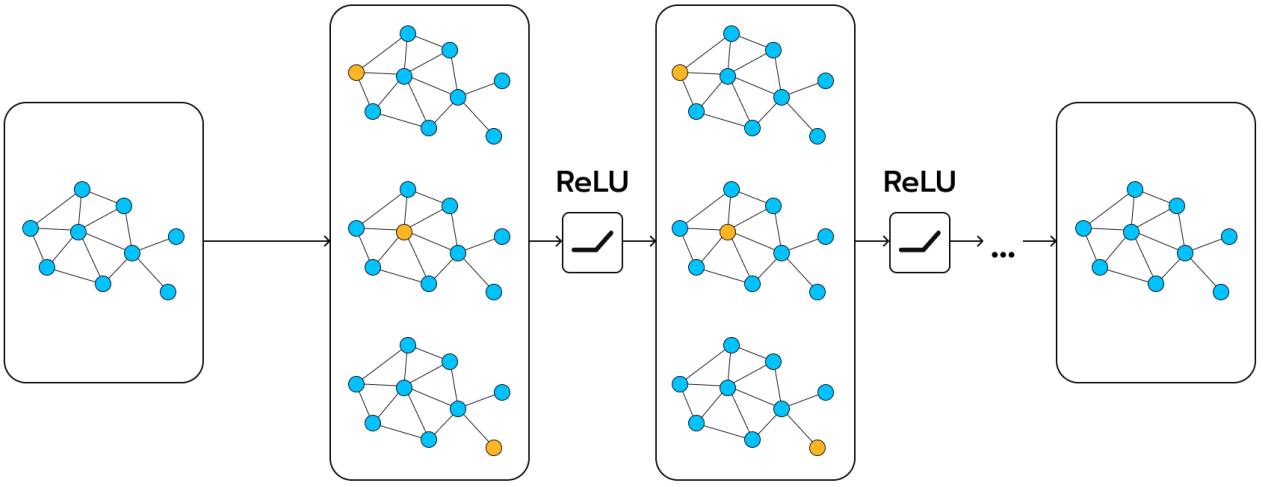


Figure 3.2: Graph Convolutional Network architecture
(Image used with permission by the author [54])

of gated recurrent units (GRUs) to control the flow of information between nodes. At each time step, the GGNN performs the following steps:

- Message Passing: Each node aggregates information from its neighbors by passing messages along the edges. The messages are computed by applying a learned transformation to the hidden states of neighboring nodes.

$$h_v^t = m_{\mathcal{N}(u)}^t = \sum_{v \in \mathcal{N}(u)} h_v^{t-1} \tag{3.9}$$

- Update Gate: The update gate determines how much of the aggregated information from neighboring nodes should be incorporated into the node's hidden state. It is typically computed using a sigmoid function, which outputs a value between 0 and 1. A value close to 0 means that the node decides to ignore the information from neighbors, while a value close to 1 means that the node fully incorporates the information.

$$u_v^t = \sigma(U_z \cdot [h_v^{t-1}] + W_z \cdot m_{\mathcal{N}(u)}^t) \tag{3.10}$$

- Reset Gate: The reset gate vector determines how much of the node's previous hidden state should be forgotten. It allows the node to selectively reset its hidden state based

on the incoming messages. The reset gate is also computed using a sigmoid function.

$$r_v^t = \sigma(\, U_r \cdot h_v^{t-1} + \, W_r \cdot m_{\mathcal{N}(u)}^t) \tag{3.11}$$

- Update Hidden State: Based on the update (u_v^t) and reset gates (r_v^t), each node updates its hidden state (h_v^t). The new hidden state is computed by combining the previous hidden state with the updated information from the aggregated messages. The update gate (u_v^t) in (3.13) determines the proportion of the updated information that is incorporated, while the reset gate (r_v^t) determines the proportion of the previous hidden state that is forgotten.

$$\tilde{h}_v^t = \tanh(\, U_u \cdot [r_i^t \odot h_v^{t-1}] + \, W_u \cdot m_{\mathcal{N}(u)}^t) \tag{3.12}$$

$$h_v^t = (1 - u_v^t) \odot h_v^{t-1} + u_v^t \odot \tilde{h}_v^t \tag{3.13}$$

W_z, U_z, W_r, U_r, W_u, U_u are all trainable and optimizable weight matrices from a neural network. By repeating the message passing, update gate, reset gate, and hidden state update steps for a fixed number of iterations, the GGNN allows information to propagate and flow through the graph structure. This enables the nodes to exchange information and capture dependencies and relationships between nodes in the graph. The final hidden states obtained after the iterations can be used for various downstream tasks, such as node classification, link prediction, or graph-level prediction. The node **v** aggregates messages from all its neighbors like in a simple message passing. Thereafter, the GRU-like update functions takes node embeddings from the other nodes and from the previous timestep to update each node's hidden state. h_v gathers the neighborhood information of node v together with information from the update and reset gate to determine the updated node v embedding. Thus, given a graph, (3.13) can be concisely written in terms of the of GRU as:

$$h_u^t = \text{GRU}(h_u^{t-1}, m_{\mathcal{N}(u)}^t) = \text{GRU}(h_u^{t-1}, h_v^t) \tag{3.14}$$

where the GRU function comprises of the update gate, reset gate and update hidden state

formulation of (3.1), (3.10), (3.11) and (3.12). v signifies all neighbors $\mathcal{N}(u)$ of each node u.

$$h_u^t = \text{GGNN}(h_u^{t-1}, h_v^t, A) \tag{3.15}$$

3.2 Relation between the Boids model and Message Passing Neural Network

The Boids algorithm, developed by Craig Reynolds in 1986 and has since been widely used in computer graphics, animation, and artificial intelligence, is a model used to simulate the flocking behavior of birds. It is used to capture the collective behavior of a flock of birds by defining simple rules that govern the movement and interaction of individual agents, or "boids," within the flock.

The behavior of each boid is influenced by three main rules: Separation, Alignment and Cohesion By combining these three rules, the Boids model can generate emergent/-collective behavior that resembles the flocking patterns observed in real birds. Individual boids respond to their local interactions corresponding to positions and movements of nearby boids, leading to self-organization and the formation of cohesive flocks.

The Boids algorithm can be implemented in a computer simulation by representing each boid as a point in space with attributes such as position, velocity, and acceleration. In each simulation step, the algorithm calculates the updated position and velocity for each boid based on the three rules and the positions and velocities of neighboring boids.

The fundamental basis of the message passing neural network is to learn and quantify how much information to aggregate from all neighbors to update each node embeddings in a graph. In each message passing operation, there is a convolution operation between node embeddings (which are agents states) and the adjacency matrix. The idea of the adjacency matrix is basically to make sure information is only exchanged with neighbors. The aggregated message is eventually combined with current node embedding to determine its next state. In the boids model, agents apply cohesion, separation and alignment forces to adjust positions and velocity in a way that mimics the flocking behavior of animals and produce collective motion patterns. The social policies for boids are given in (3.16)-(3.18)

and (3.19) defines the boid controller.

$$\mathcal{C} = \frac{1}{\tilde{\mathcal{N}}}\Big[\sum_{j=1}^{\tilde{\mathcal{N}}} p_j\Big] - p_i \tag{3.16}$$

$$\mathcal{S} = \sum_{j=1}^{\tilde{\mathcal{N}}} \frac{p_i - p_j}{||\ p_j - p_i\ ||^{n+1}} \tag{3.17}$$

$$\mathcal{A} = \frac{1}{\tilde{\mathcal{N}}} \sum_{j=1}^{\mathcal{N}} v_j \tag{3.18}$$

$$v_{j+1} = k_c\mathcal{C} + k_s\mathcal{S} + k_a\mathcal{A} \tag{3.19}$$

When the agents apply these forces: cohesion $\mathcal{C}$, separation $\mathcal{S}$ and alignment A as shown in (3.16)-(3.18), they are basically informing other agents in its neighbor their local information. In the boids model, each boid do not have global view or knowledge of the entire flock but only pat attention to local neighborhood information. This is similar to what happens in the message passing neural network where messages comprising of node embeddings of adjacent neighbors. The neural network is attempting to learn the weights to give to all contributions from other neighbors and eventually update the node. This is analogous to selecting an optimal gain parameters in (3.19) for the separation, cohesion and alignment forces.

The similarity between the Boids model and GNN message passing lies in the idea of local interactions and information propagation. In the Boids model, each boid interacts only with its nearby boids, following simple rules. Similarly, in GNN message passing, each node in the graph interacts with its neighboring nodes by passing messages, which are then aggregated to update the node's representation.

where p_i and p_j are the 2D agents i and j. p_j and v_j are the 2D positions and velocity of each agent i neighbors. $k_c, k_s$and k_a are gain vectors that weight each social policy: cohesion ($\mathcal{C}$), separation ($\mathcal{S}$) and alignment ($\mathcal{A}$). $\tilde{\mathcal{N}}$ defines the agents within a particular radius r defined as:

$$\tilde{\mathcal{N}} = j\ ||\ p_j - p_i\ || \leq r \tag{3.20}$$

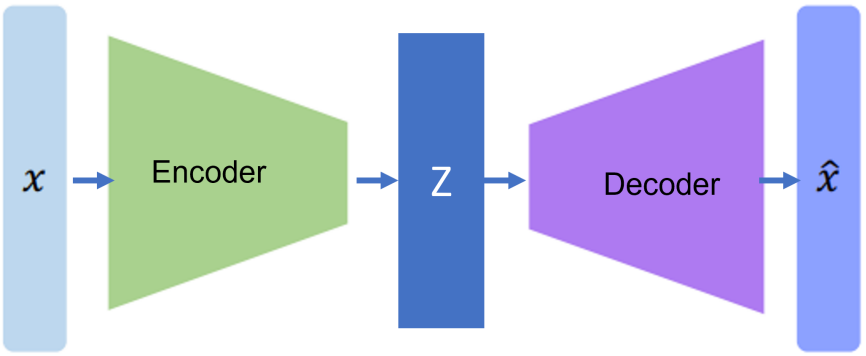


Figure 3.3: Traditional autoencoder

3.3 Traditional Autoencoders

Autoencoders are a type of unsupervised neural network that have been widely used in machine learning for tasks such as data compression, feature learning, and data reconstruction. The basic idea behind autoencoders is to learn a lower dimensional representation of the input data that captures the most important features of the data. This compressed representation is then used to reconstruct the original data with minimal loss of information. Autoencoders as shown in 3.3 consists of two main components: an encoder that maps the input data $\mathbb{X}$ to a compressed representation $\mathbb{Z}$, and a decoder that maps the compressed representation back to the original data. The encoder and decoder are trained jointly to minimize the difference between the input data and the reconstructed data.

One of the key advantages of autoencoders is their ability to learn useful features from high-dimensional data. By compressing the data into a lower-dimensional space, autoencoders can learn a more efficient representation of the data that captures its most salient features. This compressed representation can then be used for a variety of downstream tasks, such as classification, clustering, or anomaly detection. Autoencoders have been successfully applied in a wide range of domains, including image recognition, speech processing, and natural language processing. Despite their success, autoencoders can be sensitive to the choice of hyperparameters and the quality of the training data, and their performance may degrade when applied to data that is significantly different from the training data.

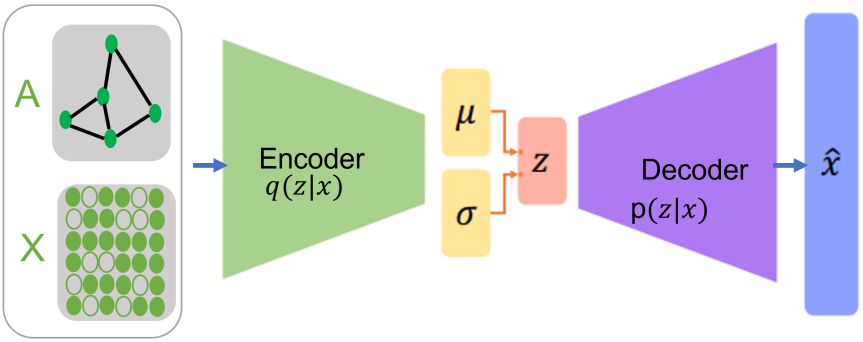


Figure 3.4: Variational graph autoencoder

3.4 Variational Graph Autoencoders

The key idea behind a Variational Autoencoder (VAE) is to learn a probability distribution over the latent variables that encode the graph. This is achieved by introducing a probabilistic layer as shown in Fig 3.4 into the encoding process, which models the latent variables as random variables with a specific prior distribution. During training, the network learns to adjust the parameters of the prior distribution such that the encoded graph representations can be accurately decoded back into the original input graph. The use of a probabilistic model in the encoding process allows the VAE to generate new graph representations by sampling from the learned probability distribution. This makes VAEs a powerful tool for generating novel data points, which can be useful for data augmentation or as a starting point for further analysis.

We want to build a variational graph autoencoder that applies the idea of VAE to graph-structured data. We want our variational graph autoencoder to be able to generate new graphs. However, we can't just straightforwardly apply the idea of VAE because graph-structured data are irregular. Each graph has a variable size of unordered nodes and each node in a graph has a different number of neighbors, so we can't just use convolution directly anymore

3.5 Bayesian Learning with Variational Inference

The Bayesian paradigm is a statistical/probabilistic paradigm in which a prior knowledge, modelled by a probability distribution, is updated each time a new observation, whose uncertainty is modelled by another probability distribution, is recorded. The whole idea that rules the Bayesian paradigm is embed in Bayes theorem that expresses the relation between the updated knowledge (the "posterior"), the prior knowledge (the "prior") and the knowledge coming from the observation (the "likelihood").

Let's assume a model where data $\mathbf{x}$ (containing position and velocity of agents in 2D) is observable and are generated from a probability distribution depending on an unknown parameter $\mathbf{z}$. $\mathbf{z}$ is a latent variable that we don't observe and that is not part of the data. Let's also assume that we have a prior knowledge about the parameter $\mathbf{z}$ that can be expressed as a probability distribution $p(z)$. Then, when data $\mathbf{x}$ are observed, we can update the prior knowledge about this parameter using the Bayes theorem as follows:

$$p(z \mid x) = \frac{p(x, z) = p(x \mid z)p(z)}{p(x)} \tag{3.21}$$

$$p(x) = \int \cdots \int_z p(x, z)\, dz_0 \ldots dz_k \tag{3.22}$$

From (3.22), it is observed that $p(x)$ is intractable and cannot be computed. Since $p(x)$ is intractable, $p(z \mid x)$ is also intractable.

Thus instead of finding $p(z \mid x)$, we find an approximating function $q(z \mid x)$ that is as close as possible to $p(z \mid x)$ and optimize by minimizing the error between the two distributions by computing KL divergence between the two probability distributions. In other words, our objective is to find:

$$q^*(z) = \underset{q(z|x) \in Q}{\arg\min} \{\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x))\} \tag{3.23}$$

Mathematically, it is the difference between entropies of probabilities distributions

$$\mathbb{D}_{KL}(Q \mid\mid P) = H_P - H_Q \tag{3.24}$$

$$\mathbb{D}_{KL}(Q \mid\mid P) = -\int_X P_X(x) \log\left(P_X(x)\right) dx + \int_X Q_X(x) \log\left(Q_X(x)\right) dx \tag{3.25}$$

The first part of (3.25) $-\int_X P_X(x) \log\left(P_X(x)\right) dx$ can be re-expressed as $-\int_X Q_X(x) \log\left(P_X(x)\right) dx$ because the KL divergence is with respect to $\mathcal{Q}$, the approximating distribution.

$$\mathbb{D}_{KL}(Q \mid\mid P) = -\int_X Q_X(x) \log\left(P_X(x)\right) dx + \int_X Q_X(x) \log\left(Q_X(x)\right) dx \tag{3.26}$$

$$= \int_X Q_X(x) \log\left(\frac{Q_X(x)}{P_X(x)}\right) dx \tag{3.27}$$

From (3.27), KL divergence between the posterior $p(x \mid x)$ and the approximating distribution $q(z \mid x)$ is given as:

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = \mathbb{E}_{q(z|x)}\Big[\log \frac{q(z \mid x)}{p(z \mid x)}\Big] = \int q(z \mid x) \log \frac{q(z \mid x)}{p(z \mid x)} d(z) \tag{3.28}$$

where $\mathbb{E}_{q(.)}$ denotes the expectation under a distribution $q(.)$ and $\mathbb{D}_{KL}$ denotes the Kullback-Leibler divergence.

From (3.21),

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = \int q(z \mid x) \log \frac{q(z \mid x)p(x)}{p(z, x)} d(z) \tag{3.29}$$

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = \int q(z \mid x) \log \frac{q(z \mid x)}{p(z, x)} d(z) + \int q(z \mid x) \log p(x) dz \tag{3.30}$$

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = \mathbb{E}_{q(z|x)} \log \frac{q(z \mid x)}{p(z, x)} + \mathbb{E}_{q(z|x)} \log p(x) dz \tag{3.31}$$

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = -\mathbb{E}_{q(z|x)} \log \frac{p(z, x)}{q(z \mid x)} + \log p(x) dz \tag{3.32}$$

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = -\mathcal{L}(q) + \log p(x) \tag{3.33}$$

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = \mathbb{E}_{q(z|x)} \log \frac{q(z \mid x)}{p(z, x)} + \mathbb{E}_{q(z|x)} \log p(x) dz \tag{3.34}$$

$$\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) = \mathbb{E}_{q(z|x)}[\log q(z \mid x)] - \mathbb{E}_{q(z|x)}p(z, x) + \log p(x) \tag{3.35}$$

where: $\log(p(x)$is the marginal log likelihood(which cannot be computed)

We note that $\log p(x)$ does not depend on q and we do not have an analytical solution for it. It should also be noted that $\mathbb{D}_{KL}$ is non-negative. Rearranging (3.35)as:

$$\log p(x) = \mathbb{E}_{q(z|x)}[p(z, x)] - \mathbb{E}_{q(z|x)}[\log q(z \mid x)] + \mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) \tag{3.36}$$

$$\log p(x) = -\mathcal{L}(q) + \mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) \tag{3.37}$$

if $\mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) \geq 0$, then we know that at minimum,

$$\log p(x) \geq -\mathcal{L}(q) \tag{3.38}$$

This is why $-\mathcal{L}(q)$ is referred to as the evidence lower bound (ELBO). Therefore, as a function of the variational distribution, minimizing the KL divergence is equivalent to maximizing the ELBO. The difference between the ELBO and the KL divergence is the log normalizer—which is what the ELBO bounds. The Evidence lower bound (ELBO), $\mathcal{L}(q)$, can be written as:

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x) - \mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z \mid x)) \tag{3.39}$$

$\mathbb{E}_{q(z|x)} \log p(x)$ is a reconstruction error of the output of the decoder which can be a mean square error or negative log likelihood. It defines how well the model is able to reconstruct the data. The KL divergence is a measure of closeness between the variational distribution and the posterior distribution. From (3.33), we see that there is still a dependence on $\log p(x)$ which is shown in (3.22) to be intractable. As a result, the objective is re-expressed in a form that can be computed shown below. Since we now know that minimizing KL is equivalent to maximizing the ELBO, we start with the expression of the ELBO.

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log \frac{p(z,x)}{q(z \mid x)} \tag{3.40}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)}[\log p(z,x) - \log q(z \mid x)] \tag{3.41}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(z,x) - \mathbb{E}_{q(z|x)} \log q(z \mid x) \tag{3.42}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x \mid z)p(z) - \mathbb{E}_{q(z|x)} \log q(z \mid x) \tag{3.43}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x \mid z) + \mathbb{E}_{q(z|x)} \log p(z) - \mathbb{E}_{q(z|x)} \log q(z \mid x) \tag{3.44}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x \mid z) + \mathbb{E}_{q(z|x)}[\log p(z) - \log q(z \mid x)] \tag{3.45}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x \mid z) + \mathbb{E}_{q(z|x)}[\log \frac{p(z)}{q(z \mid x)}] \tag{3.46}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x \mid z) - \mathbb{E}_{q(z|x)}[\log \frac{q(z \mid x)}{p(z)}] \tag{3.47}$$

$$\mathcal{L}(q) = \mathbb{E}_{q(z|x)} \log p(x \mid z) - \mathbb{D}_{KL}(q(z \mid x) \mid\mid p(z)) \tag{3.48}$$

All terms in (3.48) are known and this serves as the objective. The KL term is the negative divergence between the variational distribution $q(z \mid x)$ and the prior. Its objective is to encourage probabilities distribution that are close to the prior.

3.6 Dataset Collection and Pre-processing

This subsection presented the method utilized to obtain and preprocess the dataset for training. We also showed the steps involved in building and training the network based on techqniues previously intriduced in the previous sub-sections.

Two datasets were used for the experiments in this work: biological swarms data from schools of fish and artificial data boids model. For the biological swarms, the data was obtained from the works of [57]. Real life video motion capture data of guppies are provided and were passed through trex software [58] to obtain the observation data (position and velocity in 2D) and visual field information. To obtain the adjacency matrix, We relied on visual field information data and assume edges connection only when a fish is in the field of view of another fish. Fig. 3.5 shows the visual field information in 2D space (a projection

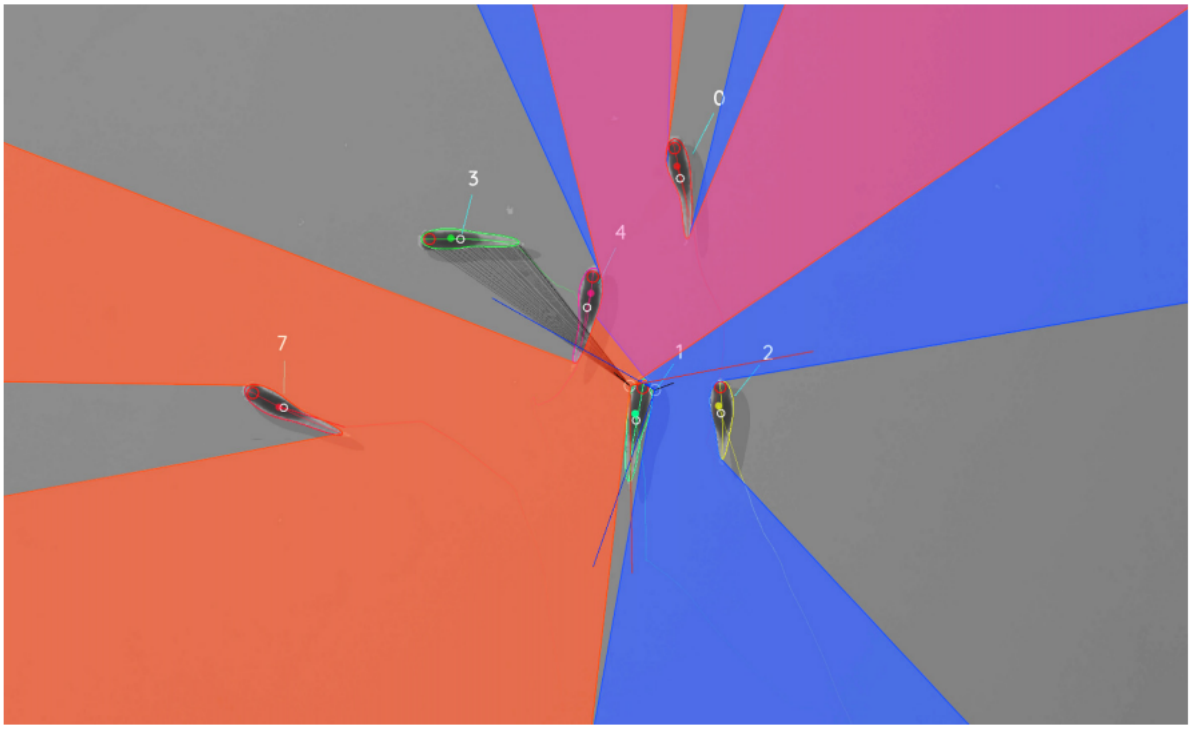


Figure 3.5: Visual field of view of Focal 1 Individual. Image used with permission by the authot [57]

from the 3d scene). In Fig 3.5, the focal individual(1), the field of view of the left eye is marked by orange, the right eye is marked by blue color and binocular vision marked by the pink region. The focal individual(1) can see most of the other fishes except 3 which is the grey region. Visible individuals are marked with ($A_{i,j} = 1$) in the adjacency matrix where i is the focal individual index and j are all the neighbors of i.

In Fig. 3.5, the Right field of the fish is represented with blue and left fields with Orange. Regions of binocular region are marked pink. The trajectories generated by the biological swarms and observations from Boids model using (3.16)-(3.19) are of the size T $\times$ N $\times$ D where T is the length of the time series, N is the number of agents in the environment and D is the dimension of the space in which the N entities exist. A graph which represents how the agents interact each other is associated with each step of a trajectory. The observations from each agent is given by the vector $[x(t), \dot{x}(t)]$ where $x(t)$ is the coordinate vector of an entity and $\dot{x}(t)$ is the velocity vector. In our experiments, an assumption that the trajectories from boids exist in a two-dimensional x and y coordinate is made, which means that $x \in \mathbb{R}^4$ for position and velocity. At each time step, the graph structure contains

the node embeddings for each node with dimension of D=4.

The dataset was divided into training, validation and test set and they were all normalized between 0 and 1 as shown in the eqn. (3.49) below:

$$x_i^{norm} = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}} \tag{3.49}$$

3.7 Architecture design

We use an unweighted graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ to describe the topological structure of the schools of fish/agent, and treat each fish/agent as a node, where $\mathcal{G}_t$ is an ordered sequence of T graph snapshots $\mathcal{G} = \mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_T$. V is a set of nodes, $\mathcal{V} = \{v_1, v_2, , v_N\}$, N is the number of the nodes, and $e_{kj}^t \in \mathcal{E}_t$ is a pair of nodes $(k, j) \in \mathcal{E}_t : \mathcal{V}_t \times \mathcal{V}_t$ and represents an edge at time step t. $\mathcal{E}_t$ is a set of edges. The adjacency matrix A is used to represent the connection between roads, $A \in \mathbb{R}^{N \times N}$. The adjacency matrix contains only elements of 0 and 1. The element is 0 if there is no link between roads and 1 denotes there is a link. The adjacency matrix is dynamic and changed every time step with respect to the field of view/occlusion of the agents.

The architecture design used to implement this work is shown in Fig. 3.6. The feature matrix $X^{N \times T \times D}$ represents the states information of the fish for all timestep T with N nodes and D dimension. At a single time step t, we have $X_t \in \mathbb{R}^{N \times D}$ where $D = 4$ representing the position and velocity in x and y direction. Generally, the node feature is the state of the fish that is observable. AT each training epoch, a graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ which can be described by the node features and adjacency matrix of $(N \times N)$ passes through the network at once, the network calculates the loss and in the next $N \times D$ timestep repeats the same process until all T graphs passes through the network for one epoch. The losses for all T for this epoch is calculated and averaged, back propagation happens and the process repeats itself for several epochs until wet of parameters that minimize error or achieve training objective is met.
The function of the encoder is to infer latent embeddings for the graph per timestep. Given input node features $x = [x_1^t, \dot{x}_1^t, x_2^t, \dot{x}_2^t]$ corresponding to the position and velocity in x and y coordinate at time t for each nodes $(N = 10)$, the encoder computes $q(z \mid x)$ through a series of mesgraph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ which can be described by the node features and adjacency

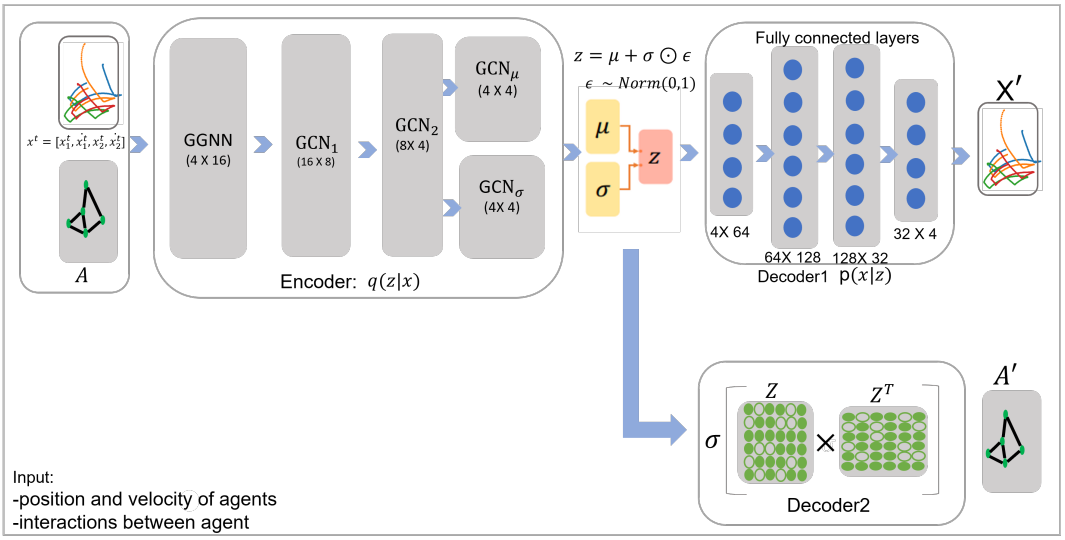


Figure 3.6: Variational Graph autoencoder architecture design

matrix of $(N \times N)$ passes through the network at once, the network calculates the loss and in the next $N \times D$ timestep repeats the same process until all T graphs passes thrpugh the network for one epoch. The losses for all T for this epoch is calculated and averaged, back propagation happens and the process repeats itself for several epochs until wet of parameters that minimize error or achieve training objective is met.sage passing operations using Graph Convolution network previously discussed in 3.1.2 for each time step $t \in T$ where z is a latent representation of the states in a graph. The input x at each time step is a $N \times 4$ vector. The other input A is an adjacency matrix of $N \times N$ of 0 or 1 entry to denote the presence or absence of edges between all the N agents The encoder network consist of a Gated Graph Neural network (GGNN) as discussed in section 3.1.3 followed by 2 Graph convolutional layers discussed in section 3.1.2. The output from the 2 GCNs is used to generate the parameters (mean μ_i and variance $\log(\sigma_i)$) of the latent embeddings. The mean and variance is then used to create a probabilistic distribution of the latent distributions.

In the architecture, GGNN has input dimension equivalent to dimension of the node feature (input dimension, D = 4) and the output dimension is chosen to be 16. In the GCN_1, the input dimension is the output dimension of the previous layer and the output dimension of GCN_2 is 16 also. In GCN_3, input dimension is 16 and output dimension is 4. GCN_μ and

GCN_σ both have input dimension of 4 and output dimension of size 4.

$$h_u^t = x_u^t \tag{3.50}$$

$$h_{v1}^{t-1} = [x_v]^{t-1} \tag{3.51}$$

$$h_{v1}^t = \text{GGNN}(h_u^t, h_{v1}^{t-1}, A) \tag{3.52}$$

$$h_{v2}^t = \text{GNN}_2(h_{v1}^t, A) \tag{3.53}$$

$$h_{v3}^t = \text{GNN}_3(h_{v2}^t, A) \tag{3.54}$$

$$\mu = \text{GNN}_\mu(h_{v3}^t, A) \tag{3.55}$$

$$\log \sigma^2 = \text{GNN}_\sigma(h_{v3}^t, A) \tag{3.56}$$

x_u^t and x_v^t are features of a node u and neighbors v respectively at time t used to initialize the input for the architecture in the GGNN layer. h_{v1}^t, h_{v2}^t,h_{v3}^t, μ, $\log \sigma^2$ are the output at each respective layers of the encoder in the architecture as indicated in (3.52), (3.53), (3.54), (3.55) and (3.56).

Decoder: The decoder uses latent representation $q^t(z \mid x)$ at each time step to compute $p(x \mid z)$ and the adjacency matrix for next timestep. There are two separate decoder models in the network architecture. The first decoder model is made up of layers of fully connected layers for predicting node embeddings $p(x \mid z)$ with the sizes of each respective layers indicated in Fig. 3.6. The output of this decoder is imitated agent states in the next time-step denoted as $X^{'}$ in the architecture design of Fig. 3.6.The other decoder is for link prediction or adjacency matrix reconstruction. Both decoders take as input the latent embeddings. For the link prediction decoder, matrix factorization using inner product is performed using node embeddings – similar nodes would basically have similar embeddings.

$$\tilde{A} = \text{Sigmoid}(Z^T Z) = \sigma(Z^T Z) \tag{3.57}$$

where Z is the node embedding matrix obtained from the output of the encoder. σ is a non linear activation function that returns a value between 0 and 1 given an input.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.58}$$

Using the dot product, nodes that are similar have the dot product to be maximal and nodes that are different have maximal dot products. Since the idea is to connect similar nodes, one is able to reconstruct the adjacency matrix using dot product to approximate each element (link) of the adjacency matrix.

3.7.1 Re-parameterization trick

The re-parameterization trick [59] is a key technique used in the training of variational autoencoders (VAEs). One of the challenges in training VAEs is that the back-propagation algorithm used to update the network parameters relies on the assumption that the latent variables are differentiable functions of the input data. However, since the latent variables are modeled as random variables with a specific prior distribution, this assumption is not true. To address this challenge, the re-parameterization trick is used to obtain a differentiable approximation of the latent variables.

The re-parameterization trick involves sampling from a standard normal distribution and transforming the samples using the mean and variance parameters learned by the network. This transformation is differentiable and allows the gradient to be back-propagated through the sampling process. By using the re-parameterization trick on the latent embeddings corresponding to the output of encoder: (mu and σ), the VAE can be trained using standard stochastic gradient descent technique.

$$Z = \mu + \sigma \cdot \epsilon \tag{3.59}$$

$$\epsilon \sim Norm(0, 1)$$

3.8 Implementation

This work was implemented with Pytorch using Pytorch framework environment with the Pytorch Geometric library [60]. Xavier Initialization [61] was used to initialize the weights

of the network by initializing the biases to be 0 and the weights Wij at each layer to be:

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right] \tag{3.60}$$

where $U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ is the uniform distribution in the interval $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ and n is the size of the previous layer (the number of columns in W).

Adam[62] optimizer was used for stochastic optimization of the VAE optimization task. The training of the model was done with 300 epochs using learning rate of 0.005. The development environment was done using JupyterLab by making use of High performance computing (HPC) GPU environment at WVU.

Chapter 4

Experiments and Result

4.1 Evaluation Metrics

Since we sample states discretely using the Boids model, the error is dependent on the sampling frequency and unot of distance used. Thus, it may become challenging to correctly estimate how the model performs. To alleviate this dependency and be able to how accurate is a step ahead prediction with, we normalize the error using the normalization factor L which is the mean average distance of the state vectors between two consecutive steps in ground truth data. The AMD (Average Minimum Distance) is an evaluation metric proposed in [63] that measures how cohesive agents in a swarm are over time. For swarms exhibiting pure focking behavior, for example, AMD should converge to a fixed value and not reach zero since it is assumed agents will not collide with each other. AMD is given by (4.4). The Average Velocity Difference (AVD) also proposed in [63] measures how aligned the velocity vectors are across all agents. For swarms exhibiting pure focking behavior, AVD is also expected to converge to a constant value. It is given by (4.5)

$$L = \frac{1}{2NT} \sum_{t=1}^{T} \sum_{i=1}^{N} \left| x_i(t+1) - x_i(t) \right| \tag{4.1}$$

$$MSE = \frac{1}{2LN(T-1)} \sum_{t=1}^{T-1} \sum_{i=1}^{N} (x_i(t) - x_i^*(t))^2 \tag{4.2}$$

$$MAE = \frac{1}{2LN(T-1)} \sum_{t=1}^{T-1} \sum_{i=1}^{N} \Big| (x_i(t) - x_i^*(t)) \Big| \tag{4.3}$$

$$AMD(t) = \frac{1}{N} \sum_{i=1}^{N} \min_j \big|\big| p_i(t) - p_j(t) \big|\big| \tag{4.4}$$

$$AVD(t) = \frac{2}{N(N-1)} \sum_{i \neq j} \big|\big| v_i(t) - v_j(t) \big|\big| \tag{4.5}$$

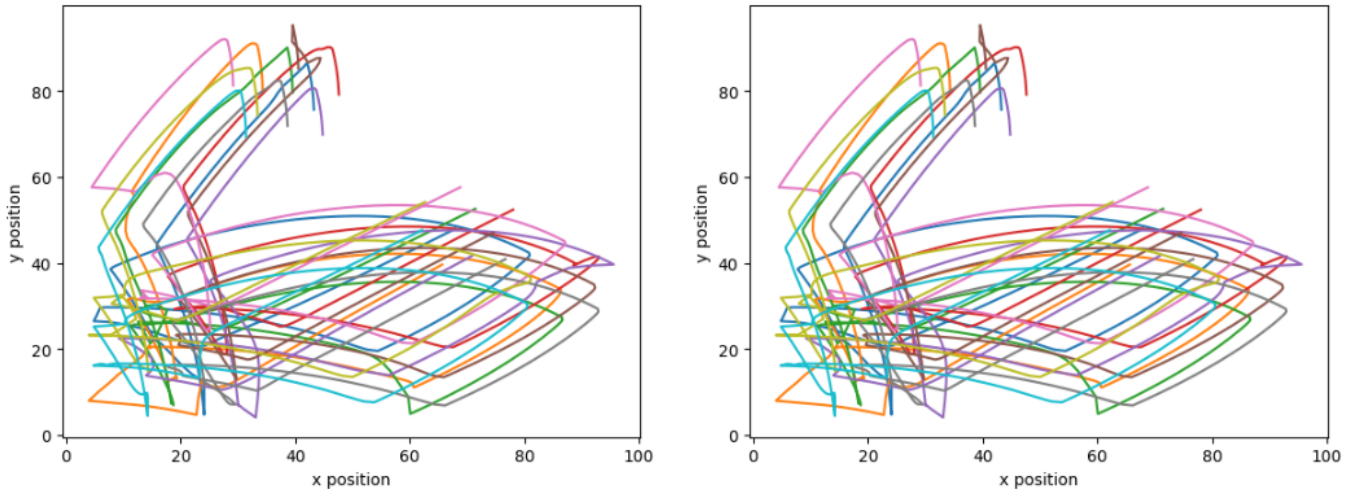
where t is the current timestep, N is the number of agents, x_i and x_j represents the observed states (position and velocity) of agents i and j respectively; and p_i and p_j are the positions of agents i and j, respectively; v_i and v_j are the velocity vectors of agents i and j.

4.2 Result with Biological Swarms and Boids

Utilizing the trained model of the VAE network architecture earlier presented in previous section, we generated new swarm configurations by sampling from the latent space. The generated swarms exhibited diverse and realistic behaviors, demonstrating the VAE's ability to learn and reproduce the complex dynamics of swarms. Experiments were done for different step predictions. This result shows and compare predicted (imitated) trajectory to actual trajectory of the agents observed for each agents. A collective trajectory for the original and imitated trajectory for all 10 boids agents is shown in Fig. 4.1a and Fig. 4.1b. Similar results were obtained for biological swarms shown in Fig. 4.2a and 4.2b. Predicted trajectory of 10 boids agent using the learnt controller is shown in Fig. 4.3 and 4.4 for the boids agents and fish respectively. During training, the MAE and MSE error for the schools of boids and fish (guppies) per epoch is as shown in Fig. 4.9 and 4.10. On the test data, the model tries to predict for K prediction steps ahead. Fig. 4.11 and 4.12 shows the MSE and MAE obtained per K prediction step.

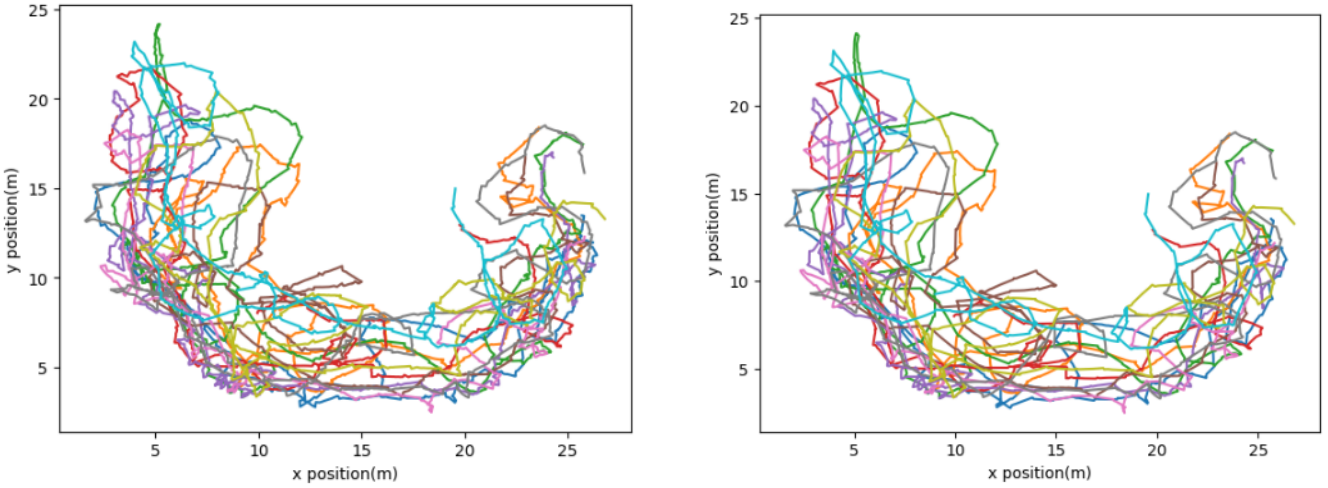
Furthermore, we evaluated the reconstruction accuracy of the VAE by comparing the original swarm observations with their reconstructed counterparts, and achieved relatively appreciable reconstructions, indicating the effectiveness of the VAE in capturing and representing swarm behavior. Fig. 4.5 shows the snapshot at different time steps of the re-

constructed graph network that represents the interactions among the agents. The numeric value of the reconstruction accuracy per timestep is shown graphically in Fig. 4.13. Table I and II shows the aggregated AMD and AVD for the boids and fish agents when evaluated on the test date split. Comparions of evaluation metric was made in the tables between technique using Graph VAE in this work, DCRNN [20] and [21] which were previously briefly discussed in the literature review.



(a) **(b)**

Figure 4.1: a: Ground Truth Boids trajectories, b: Imitating Boids trajectories



(a) **(b)**

Figure 4.2: a: Ground Truth Boids trajectories, b: Imitating Boids trajectories

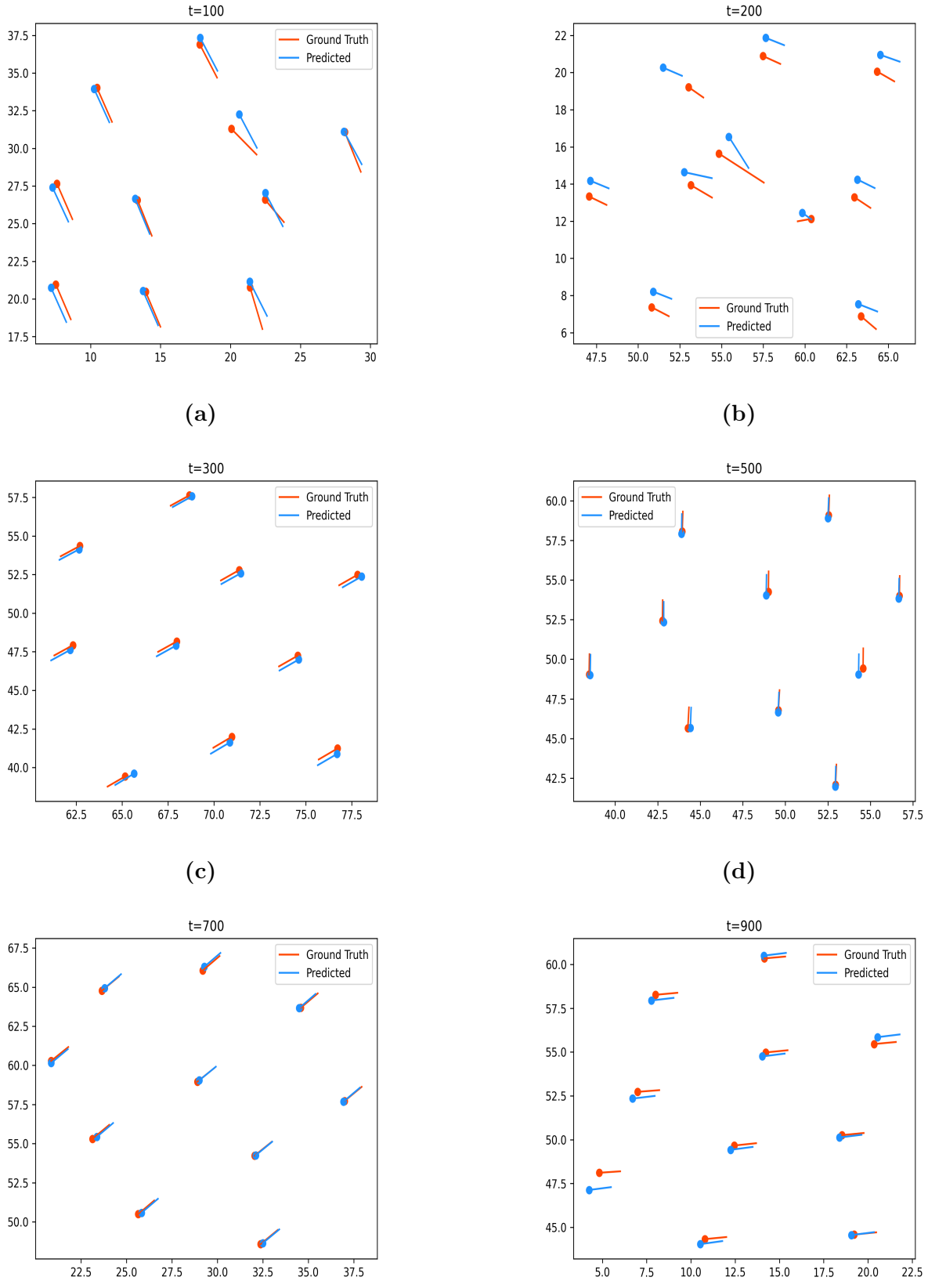


Figure 4.3: Predicted trajectory of 10 boids agent using the learnt controller. The sub-figures (a)-(f) show the snapshots of the swarm at t = 100, 200, 300, 500, 700 and 900 respectively.

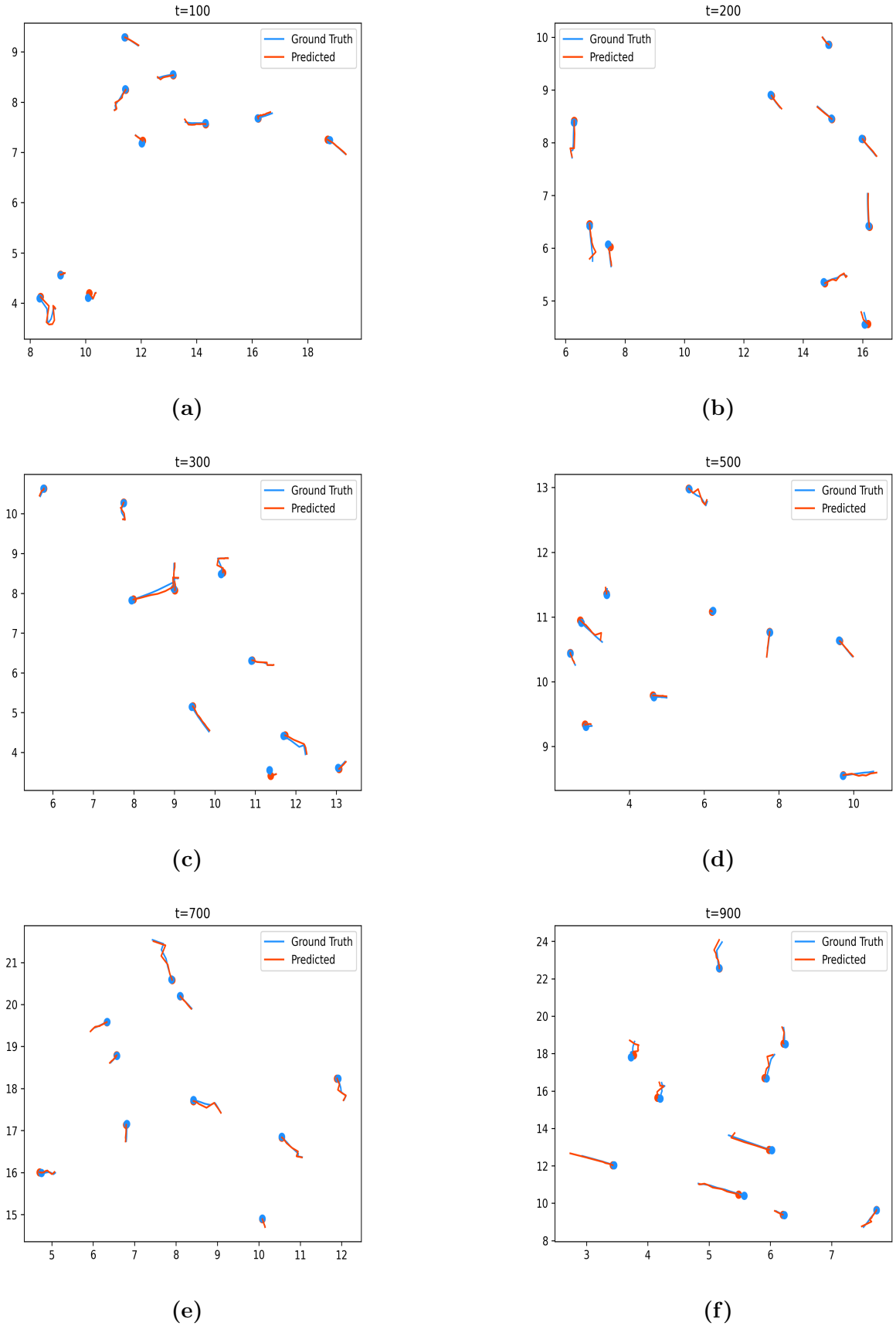
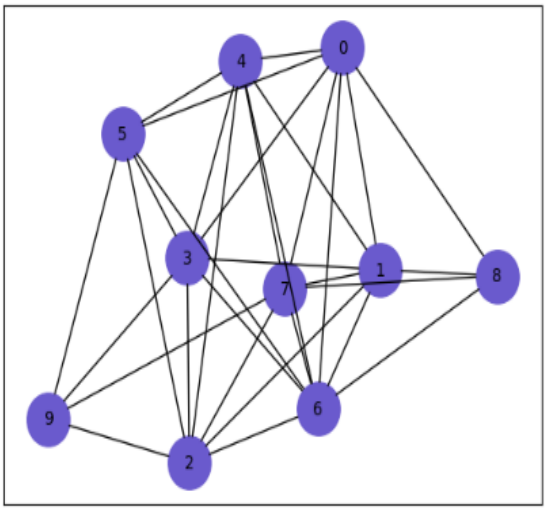
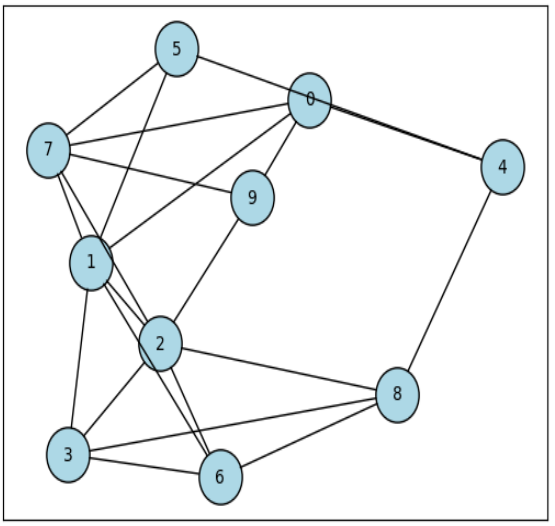


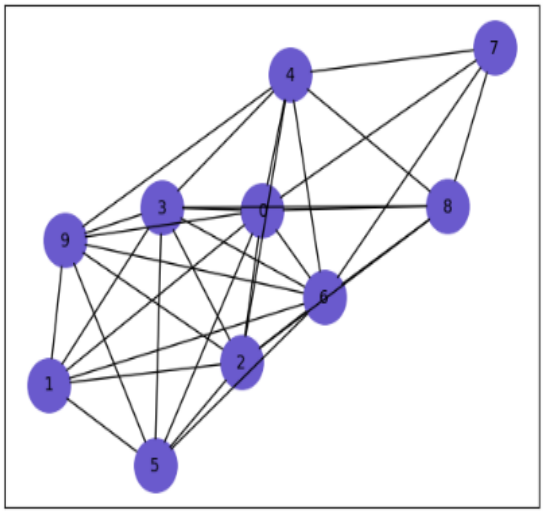
Figure 4.4: Predicted trajectory of 10 guppies agent using the learnt controller. The subfigures (a)-(f) show the snapshots of the swarm at t = 100, 200, 300, 500, 700 and 900 respectively.



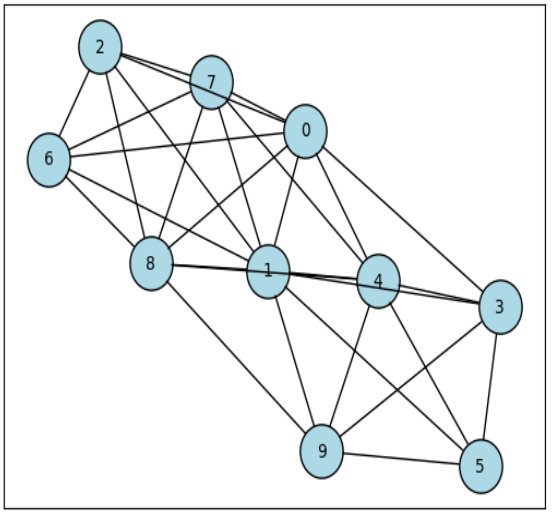
(a) Ground Truth Interactions at t=100



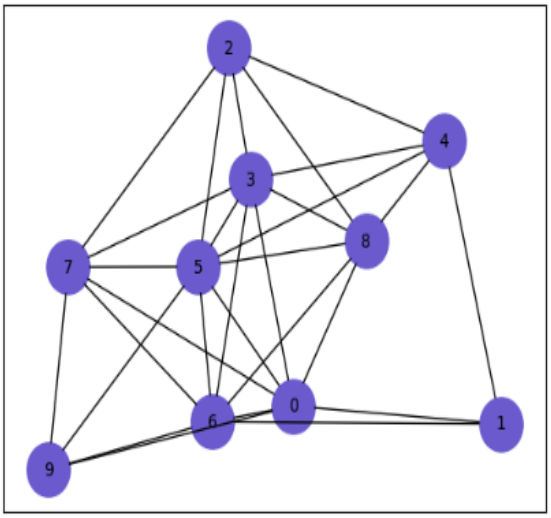
(b) Predicted Interactions at t=100



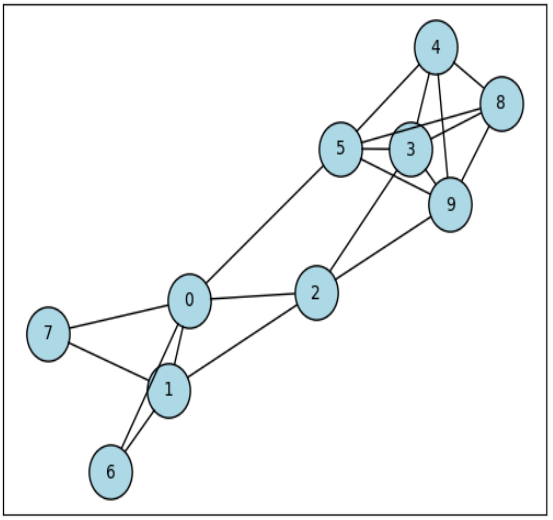
(c) Ground Truth Interactions at t=200



(d) Predicted Interactions at t=200

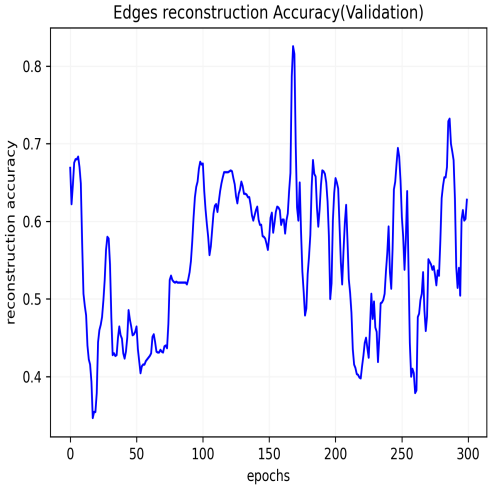


(e) Ground Truth Interactions at t=500

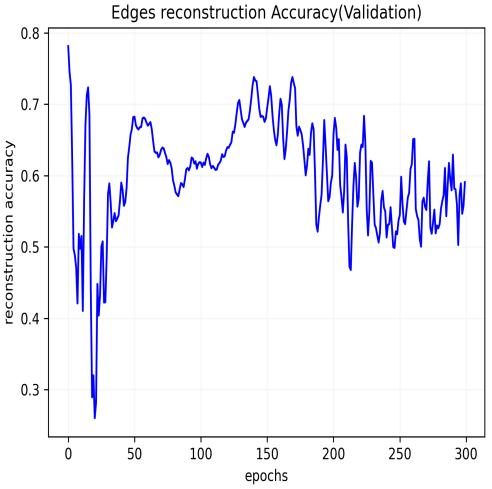


(f) Predicted Interactions at t=500

Figure 4.5: Swarm interactions for the 10 agents using the learnt controllerat t = 100, 200, and 500

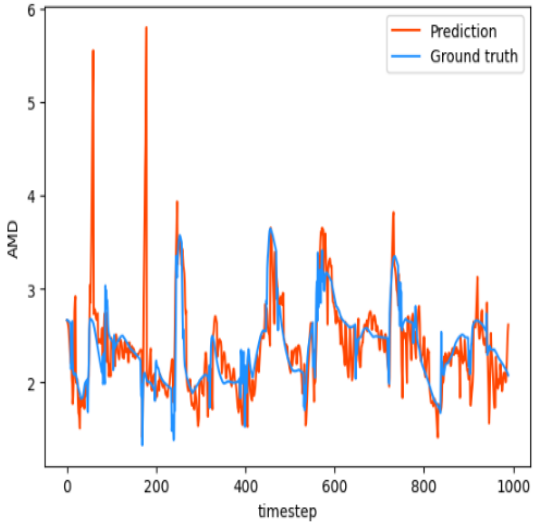


(a) Boids

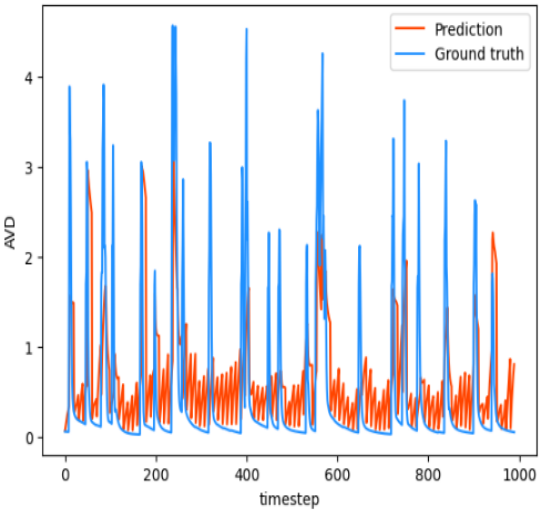


(b) Fish

Figure 4.6: Edge Reconstruction accuracy

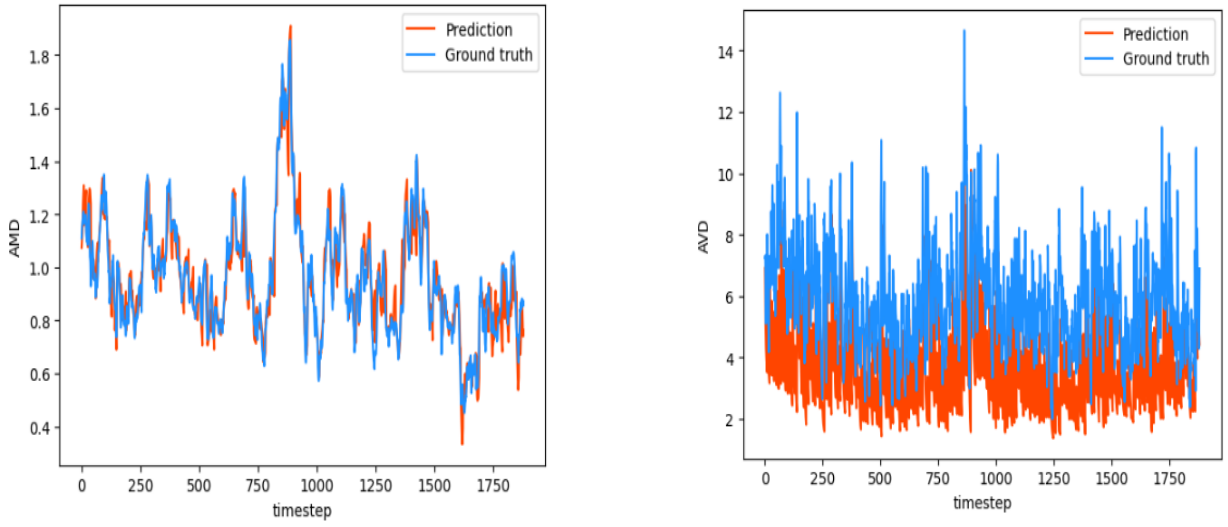


(a) AMD



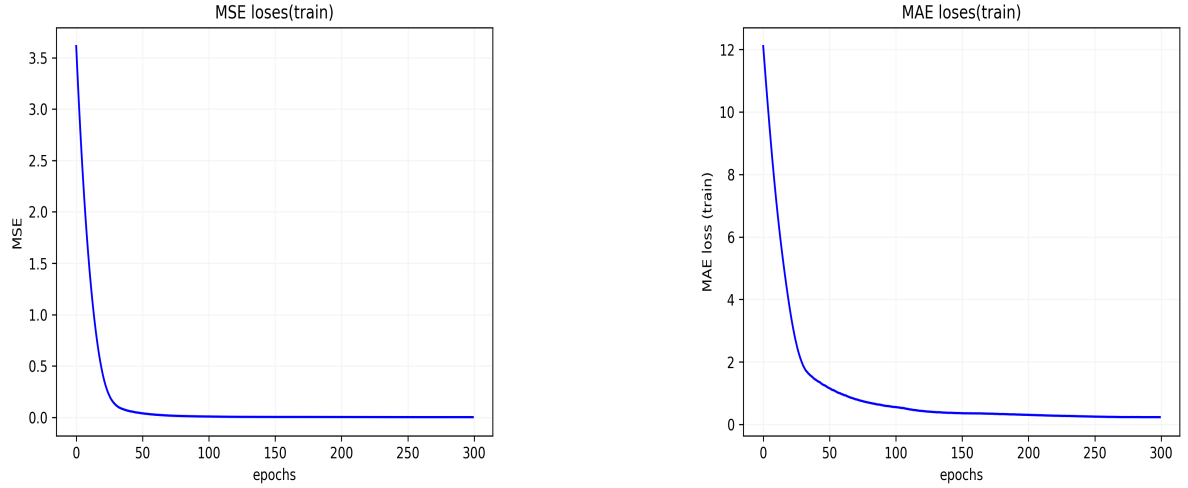
(b) AVD

Figure 4.7: Evaluation Metric on boids imitated trajectory



(a) AMD **(b)** AVD

Figure 4.8: Evaluation Metric on Guppies imitated trajectory

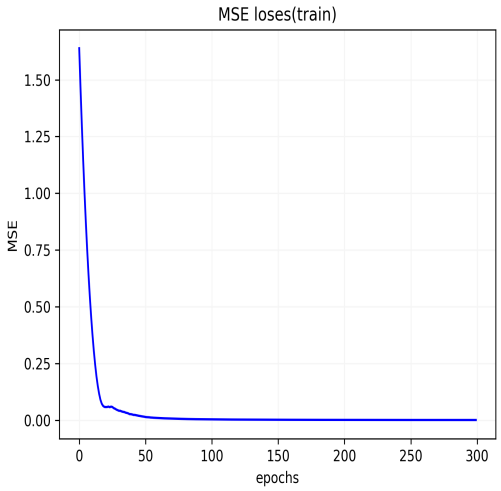


(a) MSE per epoch for Boids **(b)** MAE per epoch for Boids

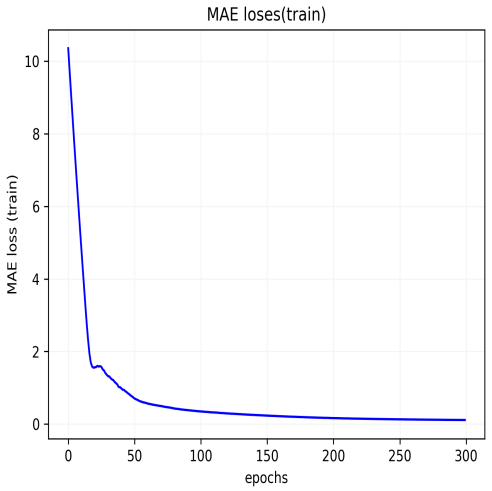
Figure 4.9: MSE and MAE per per epoch on training data for boids

4.3 Free Run Experiment

One of the main idea behind this work is to answer the question: Can agents flock given an initial position and a discovered dynamics from observations?. In order to answer this question, free run experiments was done to evaluate the observations of agents given an initial position. This model was tested on the model trained by the schools of fish data. The result obtained is shown in Fig. 4.14. The AMD and AVD obtained which measures

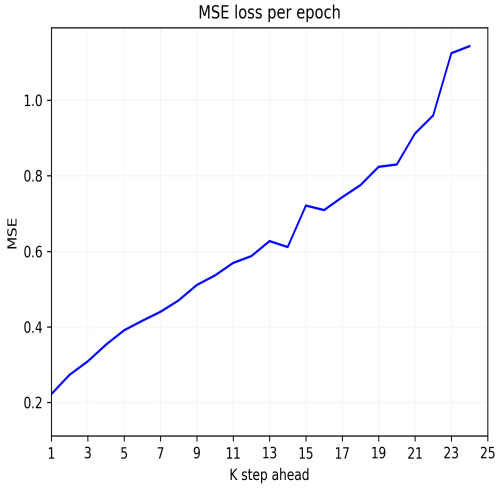


(a) MSE per epoch

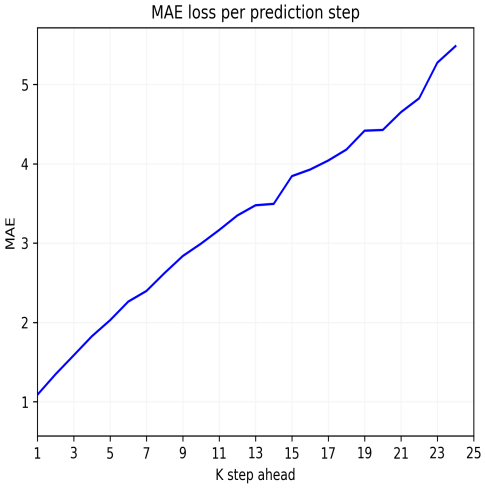


(b) MAE per epoch

Figure 4.10: MSE and MAE per per epoch on training data for Fish data



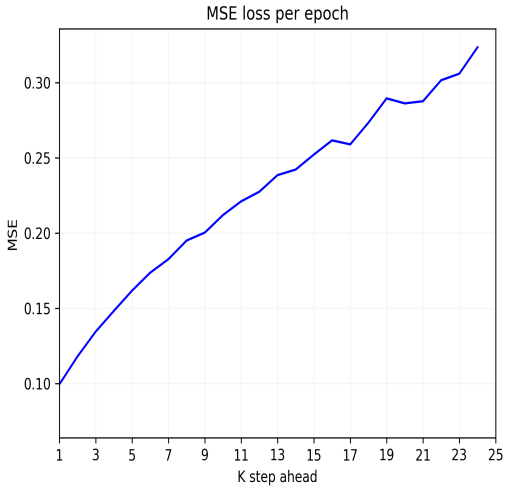
(a) MSE per prediction step for artificial swarms



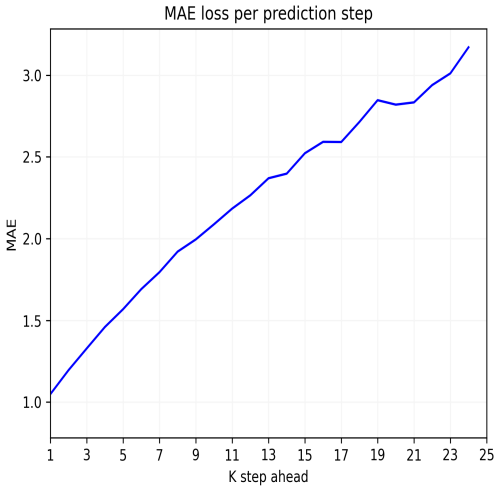
(b) MAE per prediction step for artificial swarms

Figure 4.11: MSE and MAE per Prediction steps for boids. Subfigures a-b show the MSE and MAE per on test data

how cohesive and aligned agents are with one another also indicates that there is some level of swarm behavior observed. The animation here also better represents flocking behavior observed given a random initial position.



(a) MSE



(b) MAE

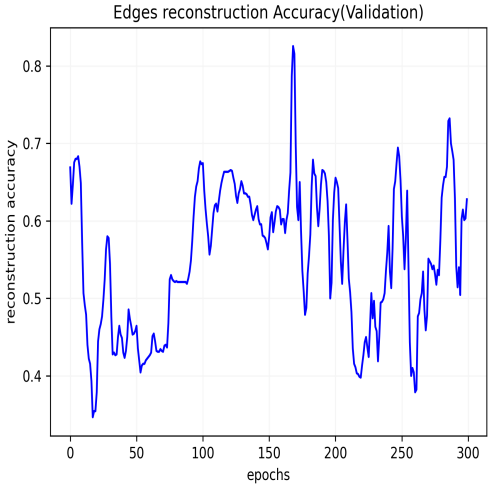
Figure 4.12: MSE and MAE per Prediction steps for guppies. Subfigures a-b show the MSE and MAE per on test data

	Graph VAE	DyGrEncoder	DRCNN
MSE	0.146	0.902	0.3452
AMD	0.96	0.957	0.902
AVD	3.47	2.451	2.5734

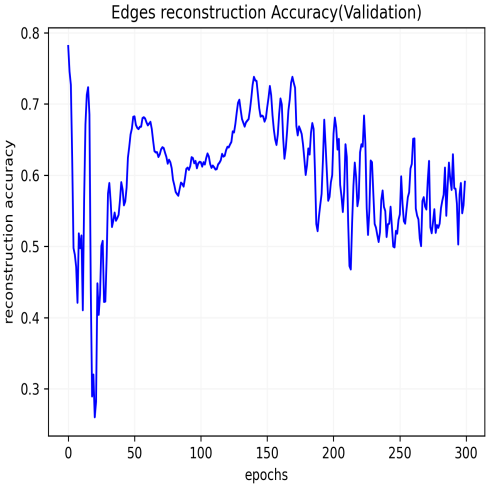
Table I: Evaluation Metric on Imitated trajectories with Guppies

	Graph VAE	DyGrEncoder	DRCNN
MSE	0.354	0.235	0.415
AMD	0.7123	0.5642	0.763
AVD	1.542	0.956	1.345

Table II: Evaluation Metric on Imitated trajectories with Boids Artificial Swarm

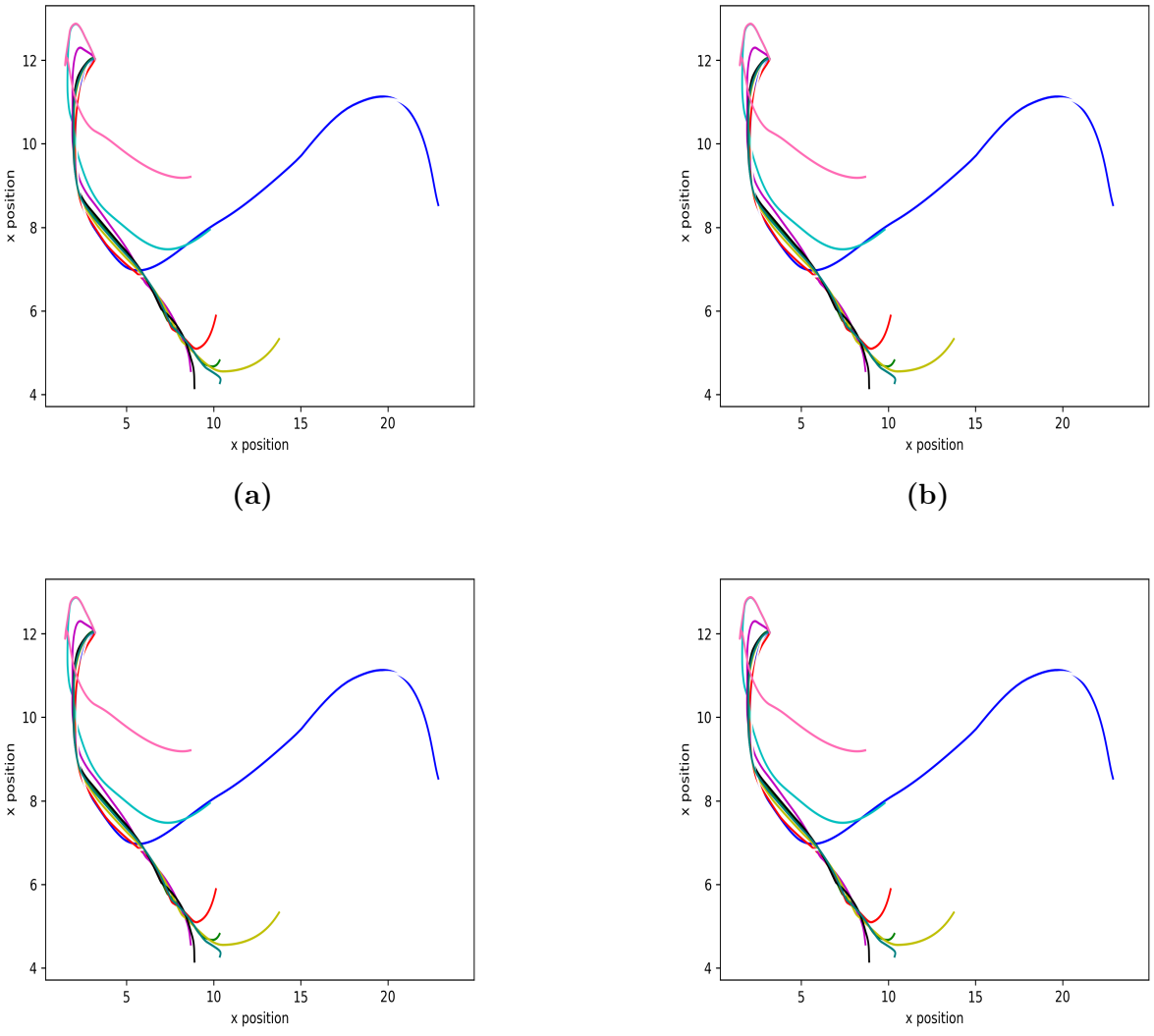


(a)



(b)

Figure 4.13: Edge Reconstruction accuracy



(a) **(b)**

(c) **(d)**

Figure 4.14: Free run simulations different initial positions

Chapter 5

Discussion and Conclusion

5.1 Discussion

A key main findings of this research is that variational inference provide a powerful framework and optimization objective for capturing and modeling the learning dynamics of biological swarms. Through the encoding and decoding processes of VAEs, we were able to extract representations of swarm behaviors and reconstruct them. This suggests that VAEs can effectively capture the underlying structure and patterns of swarm dynamics, enabling us to study and analyze their learning processes in a systematic manner. The result obtained for the AMD and AVD is able to give us a measurement of how much cohesive and aligned are the floacking properties of the agents. The MSE error evaluates the distance between the imitated trajectory and original swarm trajectory. Normalizing the MSE as discussed in the previous chapter allows us to quantify and give meaning to the MSE. We are able to understand that MSE error close to a value of 1 indicates that the imitated(next step prediction) trajectory is as close to the average of the difference between successive steps. The plots shown for the reconstructed edges in Fig. 4.13 indicates that we are able to reconstuct accuracy to a certain extent but the accuracy result obtained is not so high hence such result needs improvement possibly through other techniques or hyperparameter tuning. Importantly, the findings also shed light on the role of individual agent in collective behavior in biological swarms. It was observed that the VAE not only captured the collective behaviors of the swarm but also revealed the individual variations within the swarm. This suggests that

individual learning and adaptation contribute to the overall dynamics and emergent properties of the swarm. Understanding the interactions between individual and collective learning can provide valuable insights into how biological swarms achieve robustness, flexibility, and adaptability in response to changing environmental conditions.

The implications of the findings extend beyond the specific domain of biological swarms. The application of VAEs to model and understand complex systems has broader relevance in fields such as robotics, artificial intelligence, and social sciences. The ability of VAEs to capture the dynamics of collective behaviors and uncover underlying patterns can inform the design of intelligent systems that exhibit emergent properties and robustness which is particularly useful for imitation learning using decentralized controllers for multi-robot systems.

5.2 Limitations and Future directions

While this study has provided valuable insights into the learning dynamics of biological swarms using VAEs, there are several directions for future research that warrant exploration. Firstly, an effective way of measuring how good the result of this work is comparison of evaluation metrics with other learning methods like [35], [38]–[40], [46], [48] discussed in the literature review. This is an important step and result common to similar works in the literature and a future direction that is worthy of exploration.

Also, investigating the impact of different swarm parameters, such as swarm size, density, and communication range, on the learning dynamics would provide a more comprehensive understanding of swarm behavior. Additionally, incorporating external stimuli or environmental factors into the VAE framework could shed light on how swarms adapt and learn in response to changing conditions.

Moreover, exploring the potential of reinforcement learning techniques combined with VAEs could enable the study of swarm learning in more complex and dynamic environments. Reinforcement learning can help elucidate the decision-making processes and learning strategies employed by biological swarms, leading to a deeper understanding of their intelligence and adaptability.

Furthermore, at the moment, the encoder latent representation is not informative and

interpretable. While it is a latent representation, the idea is that the encoder should capture latent variables like cohesion force, separation force and alignment forces of boids model that determine the progression of agent states over time.

In the future, we would like to implement the controller on multiple robots to test the effectiveness of the proposed method in the real world. A promising direction is testing and adapting the GNN controller for specific application scenarios like flocking and goal seeking. This would also open up the possibilities of adding perception capabilities (such as vision) and exploring communication techniques that work well with the decentralized controller.

5.3 Conclusions

In this work, we have explored the application of variational inference techniques to multi-robot systems. Our primary goal was to develop a framework that allows robots to imitate and learn from the collective behaviors of biological swarms, with the aim of achieving enhanced coordination, adaptability, and efficiency in multi-robot systems.

Through extensive experimentation and analysis, we have demonstrated the effectiveness of variational inference in tackling key challenges in multi-robot systems. We have shown that variational methods can accurately infer the latent variables governing robot behaviors and provide robust estimates even in the presence of incomplete and noisy data. Furthermore, our proposed algorithms have proven to be scalable, allowing for the efficient inference of large-scale multi-agent systems.

One of the primary contributions of this work is the development of novel variational inference algorithms. By considering the unique characteristics of these systems, such as inter-robot dependencies and coordination, we have designed algorithms that effectively capture the joint behavior of multiple agents while still maintaining computational tractability.

While this work has made significant progress in advancing the field of variational inference for multi-robot systems, there are still several avenues for future research. One important direction is the exploration of online and distributed variational inference algorithms that can handle real-time and dynamically changing environments. Additionally, incorporating more expressive probabilistic models and designing algorithms that explicitly

capture complex inter-robot interactions could further enhance the capabilities of multi-robot systems.

In conclusion, this work has demonstrated the power and potential of variational inference in addressing the challenges of multi-robot systems. By developing novel algorithms and leveraging probabilistic modeling, we have shown that variational inference can effectively infer latent variables, handle uncertainties, and allow for imitation of flocking bahaviour observed in bioligical swarms. We hope that our findings will inspire further research in this area, leading to more robust, adaptive, and intelligent multi-robot systems in the future. The link to the code to run and repeat the experiments in this work can be found here: Link to Code on Github

References

[1] M. Yang, G.-g. Yan, and Y.-t. Tian, "A review of studies in flocking for multi-robot system," in *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, IEEE, vol. 4, 2010, pp. 28–31.

[2] Y. Liu and G. Nejat, "Robotic urban search and rescue: A survey from the control perspective," *Journal of Intelligent & Robotic Systems*, vol. 72, pp. 147–165, 2013.

[3] E. Yang and D. Gu, "Multiagent reinforcement learning for multi-robot systems: A survey," tech. rep, Tech. Rep., 2004.

[4] J. Sneyd, G. Theraula, E. Bonabeau, J.-L. Deneubourg, and N. R. Franks, *Self-organization in biological systems*. Princeton university press, 2001.

[5] C. W. Reynolds *et al.*, "Steering behaviors for autonomous characters," in *Game developers conference*, Citeseer, vol. 1999, 1999, pp. 763–782.

[6] D. Helbing, I. Farkas, and T. Vicsek, "Simulating dynamical features of escape panic," *Nature*, vol. 407, no. 6803, pp. 487–490, 2000.

[7] C. R. Kube and E. Bonabeau, "Cooperative transport by ants and robots," *Robotics and autonomous systems*, vol. 30, no. 1-2, pp. 85–101, 2000.

[8] M. Dorigo, M. Birattari, S. Garnier, *et al.*, "Swarm intelligence," *Scholarpedia*, vol. 2, no. 9, p. 1462, 2007.

[9] A. J. Sharkey, "Swarm robotics and minimalism," *Connection Science*, vol. 19, no. 3, pp. 245–260, 2007.

[10] E. Tolstaya, F. Gama, J. Paulos, G. Pappas, V. Kumar, and A. Ribeiro, "Learning decentralized controllers for robot swarms with graph neural networks," in *Conference on robot learning*, PMLR, 2020, pp. 671–682.

[11] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[12] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, PMLR, 2017, pp. 1263–1272.

[13] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," *Nature communications*, vol. 8, no. 1, p. 13 890, 2017.

[14] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, *et al.*, "Convolutional networks on graphs for learning molecular fingerprints," *Advances in neural information processing systems*, vol. 28, 2015.

[15] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[16] R. G. Baraniuk, "Compressive sensing [lecture notes]," *IEEE signal processing magazine*, vol. 24, no. 4, pp. 118–121, 2007.

[17] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: Moving beyond fingerprints," *Journal of computer-aided molecular design*, vol. 30, pp. 595–608, 2016.

[18] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[19] G. Panagopoulos, G. Nikolentzos, and M. Vazirgiannis, "Transfer graph neural networks for pandemic forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 4838–4845.

[20] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[21] A. Taheri and T. Berger-Wolf, "Predictive temporal embedding of dynamic graphs," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2019, pp. 57–64.

[22] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 922–929.

[23] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*, Springer, 2018, pp. 362–373.

[24] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," in *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*, Springer, 2018, pp. 362–373.

[25] J. Chen, X. Wang, and X. Xu, "Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction," *Applied Intelligence*, pp. 1–16, 2022.

[26] A. Pareja, G. Domeniconi, J. Chen, *et al.*, "Evolvegcn: Evolving graph convolutional networks for dynamic graphs," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 5363–5370.

[27] T. Hirakawa, T. Yamashita, T. Tamaki, *et al.*, "Can ai predict animal movements? filling gaps in animal trajectories using inverse reinforcement learning," *Ecosphere*, vol. 9, no. 10, e02447, 2018.

[28] K. Girdhar, M. Gruebele, and Y. R. Chemla, "The behavioral space of zebrafish locomotion and its neural network analog," *PloS one*, vol. 10, no. 7, e0128668, 2015.

[29] A. Khan and F. Zhang, "Using recurrent neural networks (rnns) as planners for bio-inspired robotic motion," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, IEEE, 2017, pp. 1025–1030.

[30] Z. Gao, X. Wang, B. B. Gaines, X. Shi, J. Bi, and M. Song, "Fragment-based deep molecular generation using hierarchical chemical graph representation and multi-resolution graph variational autoencoder," *Molecular Informatics*, 2023.

[31] M. Hüttenrauch, A. Šošić, and G. Neumann, "Guided deep reinforcement learning for swarm systems," *arXiv preprint arXiv:1709.06011*, 2017.

[32] G. Sartoretti, J. Kerr, Y. Shi, *et al.*, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[33] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar, "Graph policy gradients for large scale robot control," in *Conference on robot learning*, PMLR, 2020, pp. 823–834.

[34] T. Z. Jiahao, L. Pan, and M. A. Hsieh, "Learning to swarm with knowledge-based neural ordinary differential equations," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 6912–6918.

[35] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *International conference on machine learning*, PMLR, 2018, pp. 2688–2697.

[36] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[37] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[38] C. Graber and A. G. Schwing, "Dynamic neural relational inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8513–8522.

[39] S. Zhou, M. J. Phielipp, J. A. Sefair, S. I. Walker, and H. B. Amor, "Clone swarms: Learning to predict and control multi-robot systems by imitation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 4092–4099.

[40] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the national academy of sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[41] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[42] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[43] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *IEEE signal processing magazine*, vol. 25, no. 2, pp. 21–30, 2008.

[44] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *IEEE Transactions on information theory*, vol. 52, no. 2, pp. 489–509, 2006.

[45] E. J. Candes, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 59, no. 8, pp. 1207–1223, 2006.

[46] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Inferring biological networks by sparse identification of nonlinear dynamics," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 2, no. 1, pp. 52–63, 2016.

[47] K. Kaheman, J. N. Kutz, and S. L. Brunton, "Sindy-pi: A robust algorithm for parallel implicit sparse identification of nonlinear dynamics," *Proceedings of the Royal Society A*, vol. 476, no. 2242, p. 20 200 279, 2020.

[48] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019.

[49] T. Walter, A. Albi, D. Bath, *et al.*, *Reproduction Data for: TRex, a fast multi-animal tracking system with markerless identification, and 2D estimation of posture and visual fields*, version V1, 2020. DOI: 10.17617/3.4y. [Online]. Available: https://doi.org/10.17617/3.4y.

[50] D. Gedon, N. Wahlström, T. B. Schön, and L. Ljung, "Deep state space models for nonlinear system identification," *IFAC-PapersOnLine*, vol. 54, no. 7, pp. 481–486, 2021.

[51] M. Mesbahi and M. Egerstedt, "Graph theoretic methods in multiagent networks," in *Graph Theoretic Methods in Multiagent Networks*, Princeton University Press, 2010.

[52] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[53] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[54] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[55] P. Lippe, "UvA Deep Learning Tutorials," 2022.

[56] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artifical Intelligence and Machine Learning*, vol. 14, no. 3, pp. 1–159, 2020.

[57] T. Walter and I. D. Couzin, "Trex, a fast multi-animal tracking system with markerless identification, and 2d estimation of posture and visual fields," *eLife*, vol. 10, D. Lentink, C. Rutz, and S. Pujades, Eds., e64000, Feb. 2021, ISSN: 2050-084X. DOI: 10.7554/eLife.64000. [Online]. Available: https://doi.org/10.7554/eLife.64000.

[58] T. Walter and I. D. Couzin, "Trex, a fast multi-animal tracking system with markerless identification, and 2d estimation of posture and visual fields," *eLife*, vol. 10, D. Lentink, Ed., e64000, Feb. 2021, ISSN: 2050-084X. DOI: 10.7554/eLife.64000. [Online]. Available: https://doi.org/10.7554/eLife.64000.

[59] D. P. Kingma, T. Salimans, and M. Welling, "Variational dropout and the local reparameterization trick," *Advances in neural information processing systems*, vol. 28, 2015.

[60] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[61] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[63] T. Z. Jiahao, L. Pan, and M. A. Hsieh, "Learning to swarm with knowledge-based neural ordinary differential equations," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 6912–6918.