Learning Interactions and Dynamics of Swarms

Jiahao Zhang, Yeshas Thadimari, Venkat Varun Velpula, Jayanth Bhargav
University of Pennsylvania
{zjh,yeshastv,velpula,jayanthb}@seas.upenn.edu

Abstract

This paper compares and analyzes several existing methods of nonlinear System Identification (SID) for the self-propelled particles swarm model. Methods that range from simpler data-driven techniques such as Sparse Identification of Nonlinear Dynamics (SINDy) to more complex learning methods such as RNNs, CNNs and Neural ODE have been explored. The aim is to predict future trajectories of the swarm by approximating the nonlinear dynamics of the swarm model. We experiment modeling with (1) transient and (2) steady state data respectively from a swarm simulation. We demonstrate that Neural ODE, combined with a carefully selected model trained on transient data is robust to different initial conditions and can predict the correct swarm stability, outperforming other learning methods.

1 Introduction

The process of discovering and modeling interactions and dynamics of animal and robot swarms has been of interest to academics for quite some time. This topic falls under the field of nonlinear System Identification (SID). Generally swarms dynamics are non-linear and are governed by Ordinary Differential Equations (ODEs). There exists several data-driven SID methods and some of which we use as baselines which we will discuss in the Related Work section.

There exists many swarm models and there is no single unifying model that governs all swarm systems. In this paper, we focus on one type of swarm model from one particular class, which has the following general form.

$$\dot{r}_i = v_i(t), \tag{1}$$

$$\dot{v}_i = \underbrace{f(v_i(t))}_{\text{intrinsic dynamics}} - \underbrace{\sum_{j=1, i\neq j}^{N} g(r_i(t), r_j(t))}_{\text{interaction dynamics}} + \underbrace{\eta_i(t)}_{\text{noise}} \tag{2}$$

This class of swarm model is governed by a second order differential equation and the swarm behavior emerges from the acceleration, $\dot{v}_i$ as shown in equation 2. It has three parts: the intrinsic dynamics of an agent is a function of its own velocity, and its interaction dynamics is a summation of a function of its own position and other agents' positions. In addition, there is model noise which is usually assumed to be Gaussian white noise.

Models of this form can produce various steady state behaviors from flocking, milling, to aggregation. If we observe such a swarm to obtain the time series position and velocity data at a fixed sampling rate, and the form of the governing equation is unknown, we are interested in finding an approximate function that can correctly predict the trajectories given any arbitrary initial conditions.

Preprint. Under review.

Most SID methods fall into two different classes. Given a dynamical process $\dot{x} = f(x(t))$ and the initial condition x_0, one class of SID method aims to directly predict the future trajectory of x, or in other words, it aims to approximate $x_0 + \int_0^T f(x(t))dt$, which gives $x(T)$ for any T given the initial state at $t = 0$. The second class of SID methods aims to approximate $f(x(t))$, and then uses an ODE solver to numerically integrate, given the initial state at $t = 0$, to obtain future trajectories of $x(t)$. Among the methods experimented in this paper, RNN, CNN, MLP belong to the first class, while SINDy and Neural ODE belong to the second class.

Given the inherent model noise, it's impossible to perfectly predict future trajectories of all agents in a swarm, and therefore the aim of our paper is to find models that can predict the overall swarm behavior. Here, we say a model captures the correct swarm behavior if (1) it shows the correct convergence to the steady state of the actual swarm (2) it is robust to different initial conditions. In this paper we will present experiments and metrics to evaluate both requirements.

The data we use is time series data from simulations of a swarm model with the simplest ODESolver based on Euler's method. The trajectory of the system is generally split into two regimes, the transient and steady state. With randomly initialized positions of velocities, a swarm starts off in the transient state and after a certain amount of time, settles in the steady state. In this paper, we will present results of learning with data from both regimes respectively.
Since what we are training our model on is time-series data our initial candidate models included MLPs, RNNs, and CNNs.

2 Self-propelled Particles Model

We focus on one specific swarm model from the aforementioned swarm class:

$$\dot{r}_i = v_i, \tag{3}$$

$$\dot{v}_i = \underbrace{(1 - |v_i|^2)v_i}_{\text{intrinsic dynamics}} \underbrace{- \frac{a}{N} \sum_{j=1, i \neq j}^{N} (r_i(t) - r_j(t - \tau))}_{\text{interaction dynamics}} + \underbrace{\eta_i(t)}_{\text{noise}} \tag{4}$$

In this model, the intrinsic dynamics will always make an agent accelerate or decelerate to have a constant velocity of 1 as time goes to infinity. The interaction dynamics make an agent to accelerate towards the mean position of all other agents at a rate dictated by the coupling strength a. Gaussian white noise is assumed to be the same for all agents.

This swarm model exhibits three types of stability, "milling", "rotation", and "flocking". The first type of stability, "milling", happens when the model has zero time lag, low noise, and the appropriate coupling strength. Particles eventually roughly circle around the mean field with a fixed radius in a "milling" fashion. It can be the case that all particles circle in one direction, or it can also be that some particles circle clockwise and the rest circle counter-clockwise. A snapshot of this type of stability is shown in figure 1 (a). The second type of stability, "rotation", happens when significant time delay τ is introduced. The particles will eventually all aggregate and rotate in a circle with some fixed radius. A snapshot of this type of stability is shown in figure 1 (b). It's important to note that simulating swarms with Euler's method using large time steps is equivalent to introducing time delay to the swarm system. Our experiments show that, given a set of model parameters, a swarm exhibiting "milling" stability will switch to this "rotating" stability when simulation time step is increased. The third type of stability, "flocking", is when significant noise is injected to the model. Particles will all flock towards one direction. This paper focuses on the learning of systems that exhibits the first two kinds of stability.

The study of model stability is a subject in dynamical systems. We will not go into details about the proof of the model stability in this paper. Instead we rely on empirical results from our simulations to

see global convergence to stability.

This second order ODE can be written as a first order ODE. To give an example, we assume a swarm with 5 agents in total. The dynamics of each agent follows the second order ODE in equation 4. To reduce the second order ODE to first ODE, we introduce two variables for the i^{th} agent: $x_{1i} = r_i, x_{2i} = v_i$, and therefore we can rewrite the dynamics for the whole swarm as:

$$\dot{x}_{11} = x_{21}, \tag{5}$$

$$\dot{x}_{21} = (1 - |x_{21}|^2)x_{21} - \frac{a}{5}\sum_{j=1}^{5}(x_{11}(t) - x_{2j}(t)) + \eta_1(t) \tag{6}$$

$$\dot{x}_{12} = x_{22}, \tag{7}$$

$$\dot{x}_{22} = (1 - |x_{22}|^2)x_{22} - \frac{a}{5}\sum_{j=1}^{5}(x_{12}(t) - x_{2j}(t)) + \eta_2(t) \tag{8}$$

$$\dot{x}_{13} = x_{23}, \tag{9}$$

$$\dot{x}_{23} = (1 - |x_{23}|^2)x_{23} - \frac{a}{5}\sum_{j=1}^{5}(x_{13}(t) - x_{2j}(t)) + \eta_3(t) \tag{10}$$

$$\dot{x}_{14} = x_{24}, \tag{11}$$

$$\dot{x}_{24} = (1 - |x_{24}|^2)x_{24} - \frac{a}{5}\sum_{j=1}^{5}(x_{14}(t) - x_{2j}(t)) + \eta_4(t) \tag{12}$$

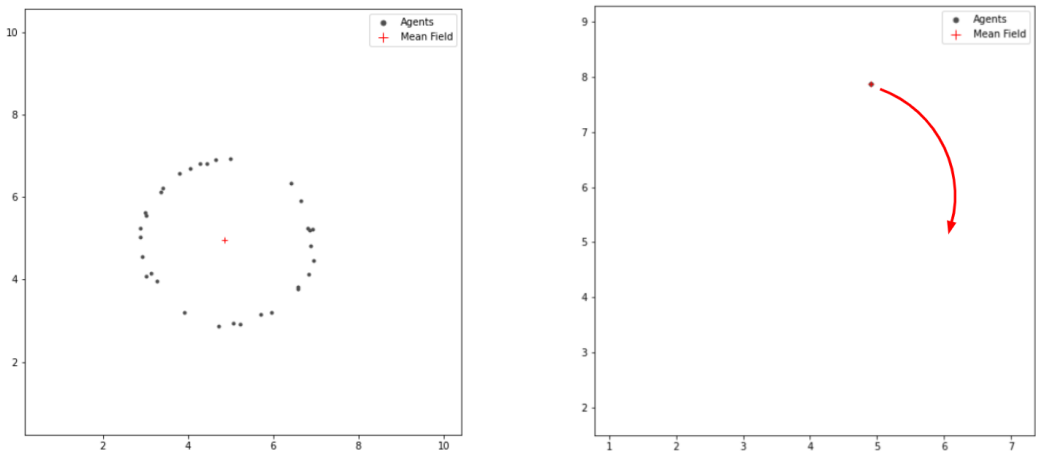
$$\dot{x}_{15} = x_{25}, \tag{13}$$

$$\dot{x}_{25} = (1 - |x_{25}|^2)x_{25} - \frac{a}{5}\sum_{j=1}^{5}(x_{15}(t) - x_{2j}(t)) + \eta_5(t) \tag{14}$$

We can then write this in vector-matrix form as:

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x}_{11} \\ \dot{x}_{12} \\ \dot{x}_{13} \\ \dot{x}_{14} \\ \dot{x}_{15} \\ \dot{x}_{21} \\ \dot{x}_{22} \\ \dot{x}_{23} \\ \dot{x}_{24} \\ \dot{x}_{25} \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{zeros}(5,5) & \mathbf{I}(5,5) & \mathbf{zeros}(5,5) \\ \hline \frac{a}{5} \cdot \mathbf{ones}(5,5) - a \cdot \mathbf{I}(5,5) & \mathbf{I}(5,5) & -\mathbf{I}(5,5) \end{array}\right] \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \\ x_{25} \\ |x_{21}|^2 x_{21} \\ |x_{22}|^2 x_{22} \\ |x_{23}|^2 x_{23} \\ |x_{24}|^2 x_{24} \\ |x_{25}|^2 x_{25} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \eta_1 \\ \eta_2 \\ \eta_3 \\ \eta_4 \\ \eta_5 \end{bmatrix} \tag{15}$$

This first order form would be the basis for our work in this paper for both simulation and learning.



(a) milling (b) rotation

Figure 1: Two kinds of stabilities exhibited by self-propelled particles. (a) shows all particles circling around the mean field, (b) shows when significant time lag is introduced all particles aggregate and rotate with a fixed radius.

3 Related Work

The collective motion of a swarm can be interpreted as combination of three principles: repulsion in short range to avoid collisions, a local interaction to align velocity vectors of nearby units and a global positional constraint to keep the flock together [1]. One particular universal feature in collective swarm motions in biological systems like a school of fish, bird flocks and a herd of mammals is that the general velocity vectors of individual are parallel to each other [2]. A study of different types of swarms is done in [3].

The topic of studying systems to discover and model interactions and dynamics of animal or robot swarms has been of interest to academics. An understanding of the dynamics can help facilitate development of better swarm systems. This process falls under the umbrella of System Identification (SID).

Identification algorithms have been developed which combine structure determination with parameter estimation using orthogonal least squares method [4]. Estimating the parameters through determining the causation entropy is useful to measure influence of components of multivariate time series data [5]. Another non-learning based method uses entropic regression to determine parameters of the dynamics equation [6].

Recently there have also been several learning-based methods for SID as well. Temporal CNNs use CNNs as a sequence model with fixed dependency range , by using masked convolutions [7]. There are also multi-step neural networks such as Adams-Bashforth [8], Cluster-Networks [9] and Physics Informed Deep Learning models [10] and [11].

Another paper, discusses the use of machine learning models to fill in gaps of knowledge-based mathematical models [12]. Generally pure analytical models can fail when for example, the noise model is not known or when it cannot be approximated.

A recent breakthrough in nonlinear system identification has resulted in a new approach to determine the underlying structure of a nonlinear dynamical system from data. This uses symbolic regression to determine dynamics and it balances the complexity of the model (number of model terms) with the agreement with data. However, the symbolic regression problem is expensive, does not clearly scale well to large-scale dynamical systems of interest, and may be prone to over-fitting. There are a host of additional techniques for modeling emergent behavior and the discovery of governing equations from time-series data. These include statistical methods of automated

inference of dynamics and equation-free modeling, including empirical dynamic modeling. Sparse Identification of Non Linear Dynamics (SINDy) is one of method developed which results in a sparse, nonlinear regression that automatically determines the relevant terms in the dynamical system from a library of functions [13].

4 Methods

Data was generated by simulations. Our data is generated from equation (**??**) in `python`. The data generated gave the position and velocities of each swarm particle for an N number of timesteps. An example of the data generated in `csv` format is given in the appendix.

The steady state of the system is defined as when the partial derivatives of the parameters of the system with respect to time is zero and will remain so. Different initial conditions of the system would mean that the system would take a different path or a different amount of time to reach the steady state. This means that for a better understanding of the system, the implementation of the methods would be split into 4 different training methodologies:

1. Training on Transient States with same Initial Conditions as Test Data
2. Training on Transient States with different Initial Conditions as Test Data
3. Training on Steady States with same Initial Conditions as Test Data
4. Training on Steady States with different Initial Conditions as Test Data

The baseline methods were implemented and tested with the above. We trained the models simply by using a sliding training window. For example, the first five steps would be used to predict the sixth step, the second to sixth step would predict the seventh step and so on. This means that a single batch considering there are 32 particles in the swarm would have the following shape:

$$\underbrace{5}_{\text{Training Window}} \times \underbrace{32}_{\text{Swarm Particles}} \times \underbrace{4}_{[x,y,\dot{x},\dot{y}]}$$

or

$$\underbrace{5}_{\text{Training Window}} \times \underbrace{128}_{[x,y,\dot{x},\dot{y}] \text{ for each swarm particle}}$$

4.1 Non Deep learning Models

In this section, two approaches are described: Regression Model based Forecasting technique to predict swarm behaviour and Functional Regression technique to learn the dynamical system equation form of the swarms.

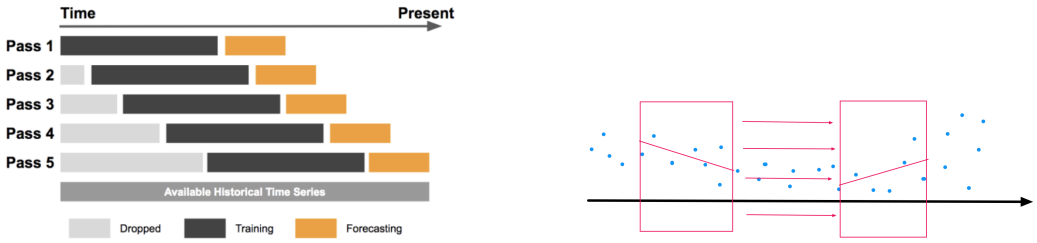
4.1.1 Least Square-Regression Model

The data we use to build models which predict future behavior of swarms is a time-series data. Generally, for simple time-series forecasting, one of the widely models are Linear Regression Models. Despite the fact that our swarm system is non-linear, linear regression models can locally estimate the non-linear function and can still be used for predicting future samples. The local neighborhood of the estimation must be quite small if we wish to achieve a good prediction. One of the first models built is a Least-Squares based Linear Regression Model. Ordinary least squares (OLS) is a method to quantify the evaluation of the different regression lines. According to OLS, we should choose the regression line that minimizes the sum of the squares of the differences between the observed dependent variable and the predicted dependent variable.
The model description is as follows:

Given a set of m samples $x = [x_1, x_2, x_3, ..., x_m]$, the model will learn a weight-vector w to predict the samples $\tilde{x} = [x_{m+1}, x_{m+2}, x_{m+3}, .., x_{m+n}]$ as $\tilde{x} = w^T x$.

The predicted samples are n-steps ahead of present time. Here m is called the number of in-samples and n is called the number of out-samples the model predicts. The choice of optimal feature set size (m) and optimal prediction horizon (n) are usually obtained by cross-validation. They are the hyper parameters for this model.

Fig.2 describes the sliding-window technique used for model building & predicting and depicts how the regression model fits lines locally on the data.



(a) Sliding window technique

(b) Locally linear - regression models

Figure 2: Time Series Regression Model using sliding window

1. **Model for Steady State Dynamics:** The swarm model data for 3000 time steps are generated using the defined model in Section 2. A test-train-validation split is created on this data. In the experiment, validation data is not really necessary. Hence only Test-Train split is created with 2000 training samples and 1000 samples for test data. The data pre-dominantly captures the steady state behaviour of the swarm system. The regression model is built which takes 10 samples and predicts 11th sample.
   After training, a sliding window technique is employed for testing the model. Initially, the last 10 samples of the training data are given to the model to predict a new sample. Now the window is slided ahead in time to include the predicted sample to predict further samples until 1000 samples are predicted.
2. **Model for Transient Dynamics:** The swarm model data for 250 time steps are generated using the defined model in Section 2. A test-train-validation split is created on this data. In the experiment, validation data is not really necessary. Hence only Test-Train split is created with 150 training samples and 100 samples for test data. The data pre-dominantly captures the transients of the swarm system. The regression model is built which takes 10 samples and predicts 11th sample.
   After training, a sliding window technique is employed for testing the model. Initially, the last 10 samples of the training data are given to the model to predict a new sample. Now the window is slided ahead in time to include the predicted sample to predict further samples until 100 samples are predicted.

4.1.2 Sparse Identification of Non Linear Dynamics (SINDy) [System-ID]

In this work, the goal is to identify the governing equations that underly a physical system based on data that may be realistically collected in simulations or experiments. Generically, we seek to represent the system as a nonlinear dynamical system

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$$

The vector $\mathbf{x}(t) = \begin{bmatrix} x_1(t) & x_2(t) & \cdots & x_n(t) \end{bmatrix}^T \in \mathbb{R}^n$ represents the state of the system at time t. These can also be regarded as the features required to learn the model. The nonlinear function $\mathbf{f}(\mathbf{x}(t))$ represents the dynamic constraints that define the equations of motion of the system.

The key findings of this learning technique is that for many systems of interest, the function f often consists of only a few terms, making it sparse in the space of possible functions. In [13], the Lorenz system which is used for analysis has very few terms in the space of polynomial functions. Recent advances in compressive sensing and sparse regression make this viewpoint of sparsity favorable, since it is now possible to determine which right hand side terms are non-zero without performing a computationally intractable brute-force search.

To determine the function f from data, we collect a time-history of the state $\mathbf{x}(t)$ and either measure the derivative $\dot{\mathbf{x}}(t)$ or approximate it numerically from x. The data is sampled at several times $t_1, t_2, \cdots, t_m$ and arranged into two large matrices: $\mathbf{X}$ and $\dot{\mathbf{X}}$. Using these two matrices, a regression model is formulated and solved.

The training data is of the form of:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \mathbf{x}^T(t_2) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \cdots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \cdots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \cdots & x_n(t_m) \end{bmatrix} \tag{16}$$

The matrix $\dot{\mathbf{X}}$ can be either given by the user (by actually recording the system derivatives) or the PySINDy module computes it by numerical techniques.

The next part of model building is the configuring the function library. The function library consists of a all candidate functions which can be present in the true system equation. We construct an augmented library $\Theta(\mathbf{X})$ consisting of candidate nonlinear functions of the columns of X. For example, $\mathbf{\Theta}(\mathbf{X})$ may consist of constant, polynomial and trigonometric terms:

$$\mathbf{\Theta}(\mathbf{X}) = \begin{bmatrix} | & | & | & | & | & | & | & | \\ \mathbf{1} & \mathbf{X} & \mathbf{X}^{P_2} & \cdots & \sin(\mathbf{X}) & \cos(\mathbf{X}) & \log(|\mathbf{X}|) & e^{\mathbf{X}} & \cdots \\ | & | & | & | & | & | & | & | \end{bmatrix} \tag{17}$$

Each column of $\mathbf{\Theta}(\mathbf{X})$ represents a candidate function for the regression problem. There is tremendous freedom of choice in constructing the entries in this matrix of nonlinearities. since we believe that only a few of these nonlinearities are active in each row of $\mathbf{f}$, we may set up a sparse regression problem to determine the sparse vectors of coefficients $\Xi = \begin{bmatrix} \boldsymbol{\xi}_1 & \boldsymbol{\xi}_2 & \cdots & \boldsymbol{\xi}_n \end{bmatrix}$ that determine which nonlinearities are active.

$$\dot{\mathbf{X}} = \mathbf{\Theta}(\mathbf{X})\Xi$$

Each column ξ_k of Ξ represents a sparse vector of coefficients determining which terms are active in the right hand side for one of the row equations $\dot{\mathbf{x}}_k = \mathbf{f}_k(\mathbf{x})$. Once Ξ has been determined, a model of each row of the governing equations may be constructed as follows:

$$\dot{\mathbf{x}}_k = \mathbf{f}_k(\mathbf{x}) = \mathbf{\Theta}\left(\mathbf{x}^T\right)\boldsymbol{\xi}_k$$

Here, $\Theta\left(x^T\right)$ is a vector of symbolic functions of elements of x, and $\Theta(X)$ is a data matrix. This results in the overall model

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) = \mathbf{\Xi}^T\left(\mathbf{\Theta}\left(\mathbf{x}^T\right)\right)^T$$

We may solve for Ξ using sparse regression. In many cases, we may need to normalize the columns of $\Theta(X)$ first to ensure that the restricted isometry property holds; this is especially important when the entries in $\mathbf{X}$ are small, since powers of $\mathbf{X}$ will be minuscule. The PySINDy package in python is used to develop these models.

1. **Model for Steady State Dynamics:** The swarm model data for 3000 time steps are generated using the defined model in Section 2. Test-Train split is created with 2000 training samples and 1000 samples for test data. The data pre-dominantly captures the steady state behaviour of the swarm system. After training the models, the results are accessed on the test data using smooth difference-differentiation function of the module. The predicted derivatives are used to evolve the swarms inital condition in the test data and is compared with the ground truth test data.
2. **Model for Transient Dynamics:** The swarm model data for 250 time steps are generated using the defined model in Section 2. A test-train-validation split is created on this data. In the experiment, validation data is not really necessary. Hence only Test-Train split is created with 150 training samples and 100 samples for test data. The data pre-dominantly captures the transients of the swarm system. After training the models, the results are accessed on the test data using smooth difference-differentiation function of the module. The predicted derivatives are used to evolve the swarms inital condition in the test data and is compared with the ground truth test data.

4.2 Deep Learning Models

For our deep learning baselines, we have tried and implemented three models. We have tested with standard feed forward network, RNNs and temporal CNNs. The latter two performed well on extrapolating on steady swarm dynamics which match well with the ground truth. But, at the same time, when trained on transient data, they failed to extrapolate and reach stability. This result shows us that these models are good at inferring temporal dependencies but are not good at learning the underlying inherent dynamics of the system.

All of the deep learning baselines were trained and tested using the methodologies mentioned in the Methods section.

4.2.1 Multi-Layer Perceptron (Fully Connected Network)

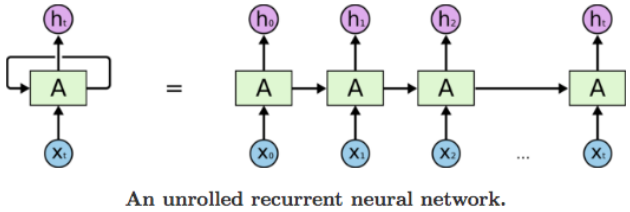
The state dynamics is an non-linear system of equations. So, first we have implemented a baseline which can possibly try capture the non-linearity of the said dynamics. A fully connected network is very useful for the same and depending on the proper activation function between the layers, could capture the temporally dependent dynamics of the swarm. The input to the neural network is a tensor of consisting of positional and velocity data of 32 particles determined for five time steps and the goal of the model is to predict the state and velocity of the swarm model for the next time-step. The below table shows the final hyperparameters that produced the most *consistent* results:

Table 1: Hyperparameters for MLP

Hidden Layer Size	256
Optimizer	Stochastic Gradient Descent (SGD)
Learning rate	0.001
Training window for dataset	5

4.2.2 Recurrent Neural Network

Recurrent Neural Networks are generalizations of feedforward neural networks with an internal memory. It is recurrent in nature as it performs the same function for every stream of input data while the output depends on the previous one computation. After an output is computed, it copied and is sent to the recurrent network. For making a decision, it considers the input and the output of the previous input. These characteristics make them very useful in recognizing patterns in time-series data or data in which the samples at at particular time can be assumed to be dependent on the sample previous to it.



An unrolled recurrent neural network.

Figure 3: Unrolled Structure of RNN
Reference

We have used a single layer RNN layer before passing the output through a linear activation function. The loss function used was `nn.MSELoss()`. The hyperprameters tuned were the hidden size, number of layers of RNN, the optimizer, learning rate. The hyperparameters are as follows:

4.2.3 Convolutional Neural Networks

While RNNs and its derivatives like LSTMs, GRUs were the de facto models to use when dealing with time series based data, CNNs have also been widely used in the learning of time series data as

Table 2: Hyperparameters for RNN

Hidden Layer Size	256
Number of RNN layers	1
Optimizer	Stochastic Gradient Descent (SGD)
Learning rate	0.005
Training window for dataset	5

explined in [14]. Since there are several swarm particles in the data and all with underlying interaction forces, having a form of convolution may prove to be beneficial in learning the system model, i.e. the convolution may be beneficial in capturing local information as well as the temporal information. In this implementation, 1-Dimensional convolutions were used instead of 2-Dimensional convolutions. Below is the final architecture of our implemented model:

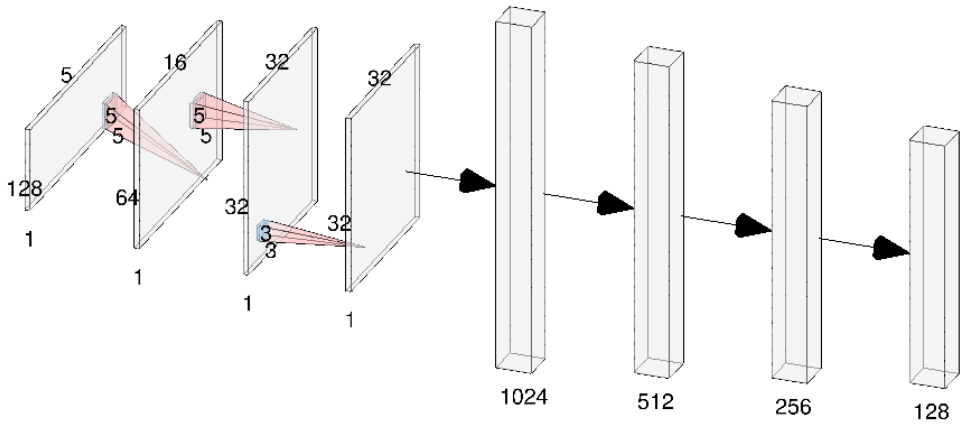


Figure 4: Implemented CNN Architecture

Several architectures and filter sizes were experimented with but this gave us the most consistent results. Max-Pooling, Adaptive Average-Pooling and Dropout were also experimented with, but they produced rather poor results. The chosen activation function used was `ReLU` for most layers. Once again, the chosen loss function was Mean Square Error (`nn.MSE`).
The main optimizers experimented with were Stochastic Gradient Descent (SGD) and Adam. While Adam would sometimes quickly converge, it was inconsistent at various learning rates. However, SGD would consistently converge quickly at a learning rate of 5×10^{-3} and was these were the final chosen optimizer and learning rate.

Table 3: Hyperparamters for CNN

Optimizer	SGD
Loss Function	Mean Squared Error
Learning Rate	5×10^{-3}
Training window for dataset	5

Table 4: Parameters of Convolution Layers

First Layer		Second Layer		Final Layer	
Filter_1	5	Filter_2	5	Filter_3	3
Padding_1	2	Padding_2	2	Padding_3	1
Stride_1	2	Stride_2	2	Stride_3	1

These models do not perform well when it came to prediction and generalization on transient models. These results paves a way towards a search of deep learning solutions which are more robust and are able to generalize the dynamics and are able to better predict future trajectories of swarm particles when undergoing transient dynamics.

4.3 Advanced Deep Learning Model - Neural ODE & Physics-informed Architecture

Among all baseline models tested, SINDy performed best in terms of learning from transient data, and therefore learning $f(x(t))$ instead of $x(t)$ becomes a natural choice for advanced model. [] proposed Neural ODE for learning dynamics through optimizing over ODESolvers by defining adjoint states. This means that the learnt model have a continuous latent space and allows the neural network to leverage physical assumptions of a swarm model.

This advanced model has two components that make it unique and powerful. First, it uses the idea of Neural ODE, which is itself not an architecture, but a novel way to perform optimization in the presence of an ODE Solver. Backpropagating through an ODE Solver is possible but it requires the storing variables at each time step and therefore incurs high memory cost. Instead it defines adjoint states whose time dynamics can be obtained by numerical integration with the ODE Solver. These adjoint states can then be used to calculate the derivatives of each parameters with respect to the loss function. In other words the ODE Solver is used as a black box and it does not matter what kind of ODE Solver is being used.

Given the form in equation 15, the goal is to approximate the function $\dot{\mathbf{X}} = f(\mathbf{X})$ with a neural network $\hat{f}$ with parameters θ, and given a scalar-valued loss function L, then our primary objective is to minimize $L(\mathbf{ODESolve}(\mathbf{X}(t_0), \hat{f}, t_0, t_1, \theta))$. The definition of adjoint states and its time dynamics calculation can be found in the original paper [15].

The second component of this model is the choice of neural network architecture used to approximate the function $\hat{f}$. Since we are directly approximating the dynamics $\dot{\mathbf{X}}$, we can use some physical assumptions of the swarm system to inform the choice of neural networks. Given that the observed swarm is homogeneous, we know that the dynamics of each agent must be the same, and the ways an agent interacts with other agents should be the same. Based on this, we designed the architecture as shown in figure 5. Using a physics-informed network significantly reduces parameter space and therefore leads to faster training.

Table 5 and 6 shows the structure of final architecture of the aforesaid model and the hyperparameters used for training.

Table 5: Physics-informed model architecture

	Intrinsic Block	Interaction Block	Aggregation Block
Number of layers	3	3	3
Nonlinearity at input	cubic terms	None	None

Table 6: Physics-informed model hyperparameters

ODESolver type	ODESolver step size	Optimizer	Loss Function
Euler's	0.05	Adam (lr = 0.001)	MSE

Another variant of this model explicitly includes model noise in the neural network. With the assumption that model noise is normally distributed, we can introduce bottlenecks as shown in figure 6. The sampling block reparametrizes from the mean and variance to add noise to the output of each agent. While this model did not perform better than the previous model, it showed faster convergence in training.

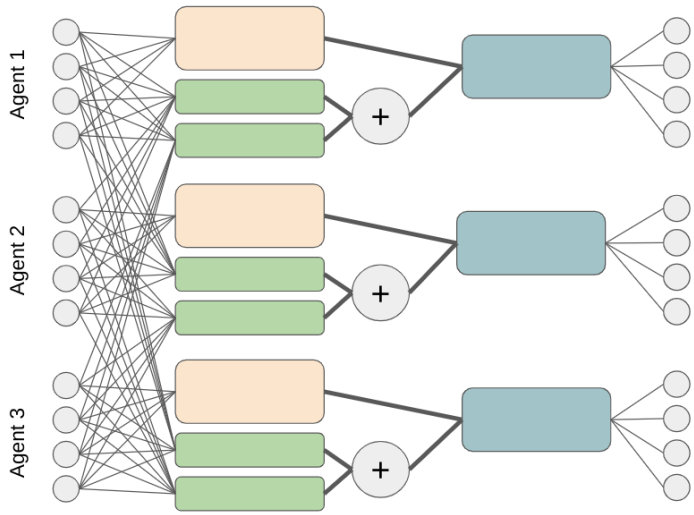


Figure 5: Physics-informed model for $f(x(t))$ in a 3-agent swarm, all pink blocks are identical fully connected layers acting on each agent itself (intrinsic), all green blocks are identical fully connected layers approximating the interaction between two agents (interaction), and all blue blocks are fully connected layers processing the intrinsic and interaction information from previous layers.

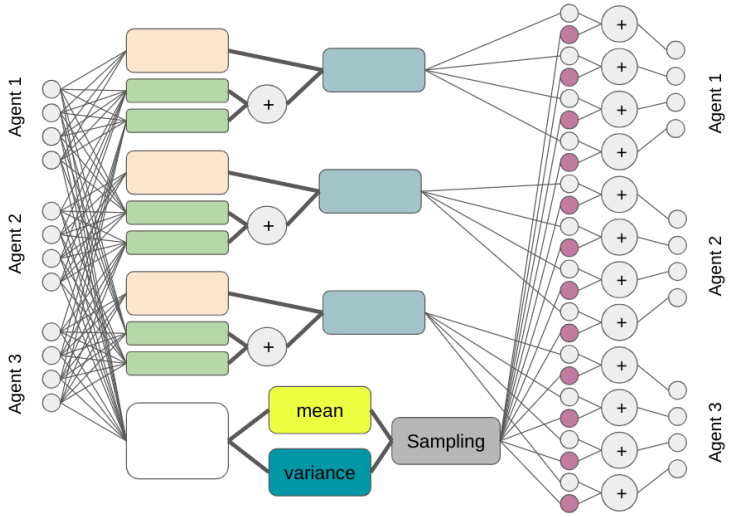


Figure 6: Physics-informed model for $f(x(t))$ with noise modeling. This is the same as the model in figure 5 except that it explicitly models the noise profile and sample noise from the modeled distribution to add to the output.

5 Analysis

The metric we came up with to compare our models is the *Mean Field Error* (MFE). This is basically the error of the mean positions of all predicted swarm particles versus the ground truth particles. Essentially it is the euclidean distance between the mean positions of all swarm particles. This would give an intuition if there was any form of stability reached and if the model was able to learn from the time-series data. The Mean Field Error is computed as follows:

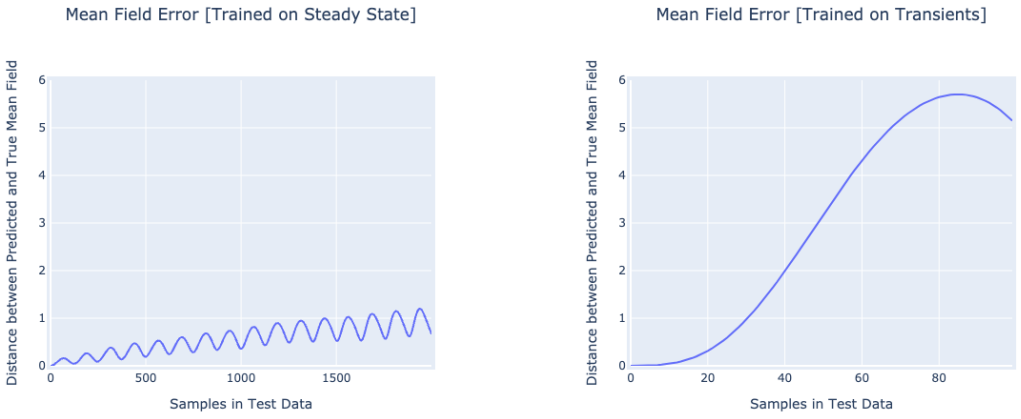
$$MFE = \sqrt{(x_{m,true} - x_{m,pred})^2 + (y_{m,true} - y_{m,pred})^2}$$

5.1 Non Deep Learning Baselines

5.1.1 Least Square Regression Model

The time-series regression model is evaluated on test data for both steady state and transients of the swarm.

1. **Steady State Model:** The model predicts 2000 time steps ahead of the training set which is compared with the corresponding test data set (ground truth). The mean field of the ground truth and predicted trajectories for 2000 steps are observed and the mean field error is visualised. The steady state model is trained and tested on a 32-agent system.
2. **Transient Model:** The model predicts 100 time steps ahead of the training set which is compared with the test data set (ground truth). The mean field of the ground truth and predicted trajectories for 100 steps are observed and the mean field error is visualised. The transient model is trained and tested on a 10-agent system.



(a) MFE for Steady State Predictions

(b) MFE for Transient Predictions

Figure 7: MFE for Linear Regression Model

It can be concluded that, with an optimal feature size and prediction horizon of 10 and 1 respectively, the regression model is able to predict the steady state evolution of the swarm with minimal MFE. However, with increase in the number of samples of prediction, the MFE starts increasing. Therefore, the regression model is able to predict the steady state behaviour of the swarm for around 1000 steps ahead with acceptable MFE. On the contrary, the models performance on transient data is not appreciable. The model is not capable of capturing the highly non-linear transients of the swarm system.

Also, such a regression model does not give any insight into the hidden dynamics of the swarm system since by nature the regression is linear and the swarm model is highly non-linear. A plausible reason as to why the regression model predicts well on steady state would be that the steady state stability of this multi-agent system is linear in nature and the model is able to capture this.

5.1.2 Sparse-ID (SINDy) Functional Regression Model

The case studies considered for SINDy based models are:

i. Training on Steady State Dynamics data and testing the predictions

ii. Training on Transient State dynamics data and testing the transient predictions

iii. Using Transient models to extend the predictions and observe wether they are able to reproduce steady state trajectories and learn the true steady state stability of the swarm system.

Case Study 1: Training on Steady State Dynamics data and testing the predictions

The 32-agent swarm data is used to train this model. 2000 training samples and 1000 testing samples are used to build and evaluate the model. The mean-field error of the model predictions on test data is plotted over the samples in test data. It can be observed that the model is able to predict the steady state with minimal MFE. This model is built with the default library of the PySINDy package without customizing or changing the library functions. It can be concluded that, with polynomial functions, the SINDy model is able to learn the steady-state evolution of the swarms with high accuracy. However, as prediction horizon increases, the MFE also starts to increase. The MFE plot for this case study is in Fig. 8a.

Case Study 2: Training on Transient State dynamics data and testing the transient predictions

A 10-agent swarm data is used to train this model. 150 training samples and 100 testing samples are used to build and evaluate the model. The mean-field error of the model predictions on test data is plotted over the samples in test data. It can be observed that the model is able to predict the steady state with minimal MFE. Since the transients of the swarm system are highly non-linear and chaotic in nature, several different choices of functional libraries are considered for building models.
The function libraries which are considered for this case-study are:

i. Polynomial Library (upto 2nd order)

ii. Polynomial Library without Interaction terms

iii. Fourier Library

iv. Custom Library 1 (without interaction functions)

v. Custom Library 2 (with interaction functions)

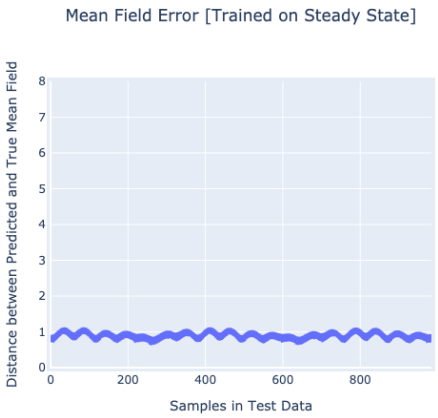
The MFE plots for this case study are in Fig. 8b to Fig.8f.

Case Study 3: Using Transient models to extend the predictions and observe whether they are able to reproduce steady state trajectories and learn the true steady state stability of the swarm system.

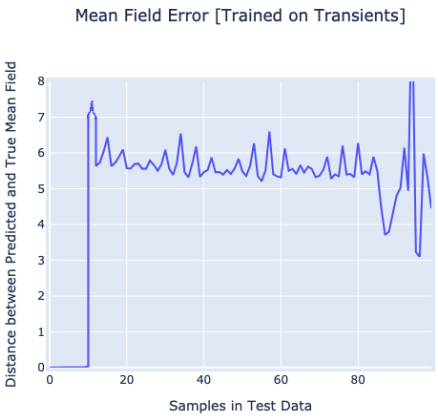
A model trained on the transients of 32 agent system is used to evolve the last training sample as a initial state to predict samples ahead of training data set. These predictions are used to analyse if the model is able to reproduce steady state trajectories and learn the true steady state stability of the swarm system.

In the absence of ground truth, the only metric to analyse the model performance is the formation of the swarms in the predicted time range. It is observed that the swarms, individually, move in circles with different centers. This is some sort of periodic motion and hence a stability but not the true stability as expected from the system. This behaviour could possibly be exhibited because the model was not able learn the interactions but only the stability.

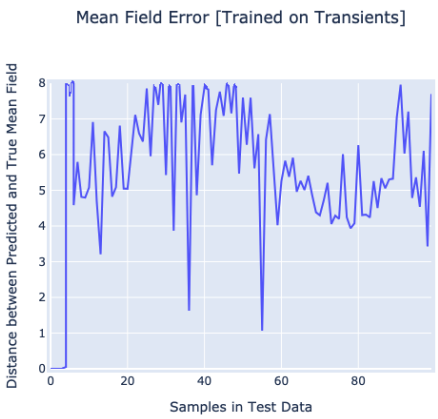
From the MFE plots and swarm behaviour as described in Fig.8 and Fig.9, it can be concluded that SINDy models learn well on the steady state data but not on transients. However, the model with function library containing interaction terms/functions (Custom Library 2) has the least the MFE amongst the other libraries. Although the identified model produces derivatives that accurately match the measured derivatives, the dynamic model does not agree with the true system, except for a very short time at the beginning of the simulation. As the the horizon of prediction increases and more samples are predicted, the error shoots up. The best of the SINDy models are capable of only approximating the models for shorter horizons of forecast but fail to generalize well on different initial conditions and longer prediction horizons. Fig.9 shows some frames of the swarm motion. The system exhibits a periodic motion where all agents move in circles with centres as in Fig.9b and thus this proves that the model has learnt the stability but not the true stability which has interactions.



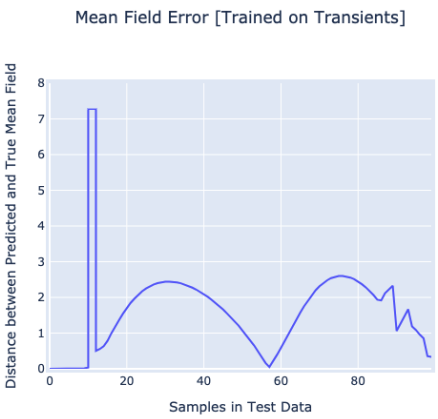
(a) Steady State Predictions (SINDy)



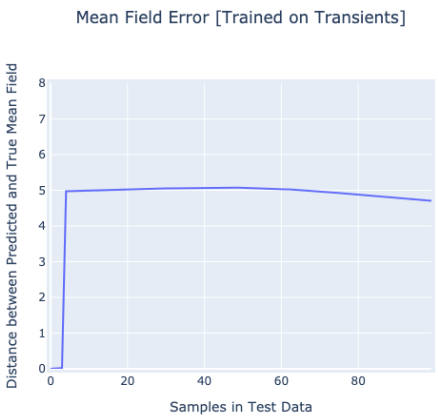
(b) Polynomial Library



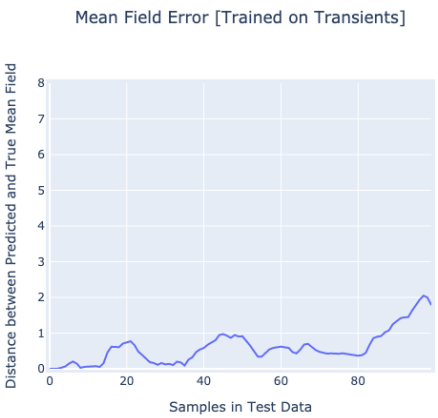
(c) Polynomial Library (Non interactions)



(d) Fourier Library



(e) Custom Library 1



(f) Custom Library 2

Figure 8: MFE Plots for SINDy Models

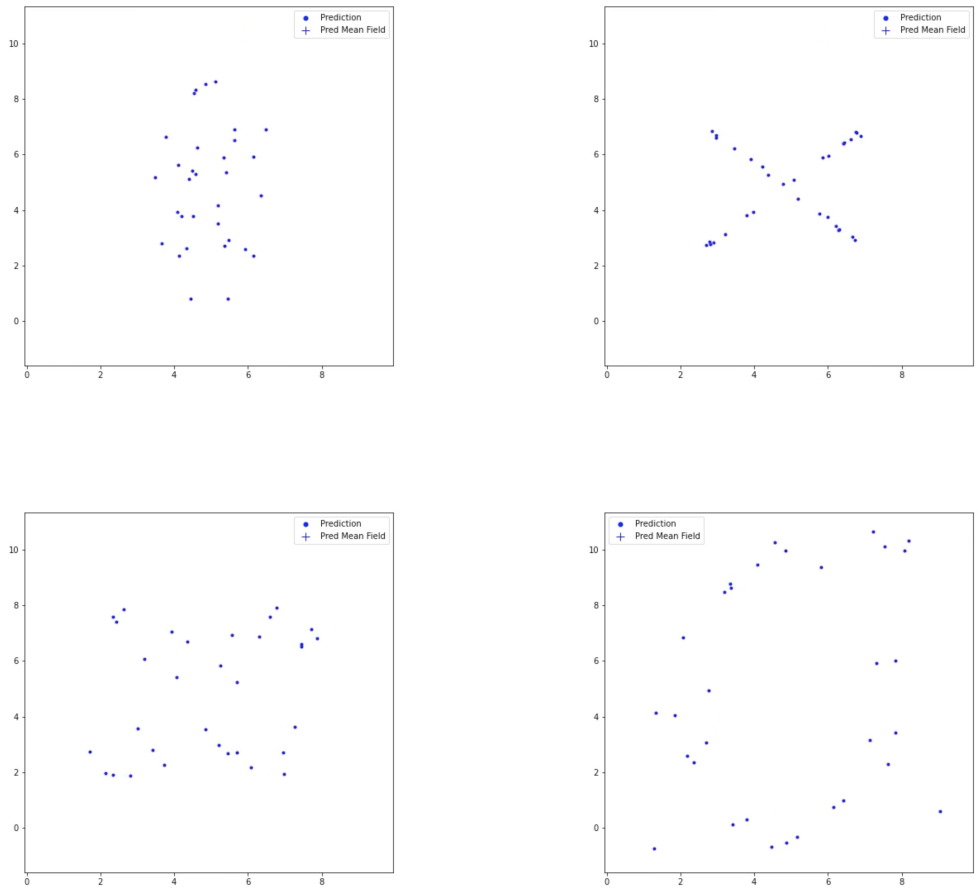


Figure 9: Prediction frames on steady state of SINDy Model trained on transients

5.2 Deep Learning Baselines

The first model tested was the Multi-Layer Perceptron (MLP) model. The MLP was able to learn the non-linearity property of the model. But the performance of the model highly dependent on initial conditions of the particles. We noticed a significant drop in performance (ability to predict the true test trajectories of the swarm) when we the swarm starts from a different initial condition.

Next we have compared the performance of the model on steady state data versus the data consisting of both transient dynamics and steady state dynamics. We saw significant drop of performance in the latter case when the model is made to predict future actions, especially during the transient phase. This can be seen explicitly by the mean field errors.

These results show that the MLP is not able to learn the hidden characteristics of the swarm behavior and simply trying reduce mean field error. Hence it would fail when the initial conditions become different or during transient cases. Another disadvantage of MLP over the other baselines is the amount of training time. The model was trained for 500 epochs compared to the number epochs RNN and CNN models trained for (50 epochs). Due to the inconsistencies of MLP, it was not tested with as extensively as the CNN and RNN model.

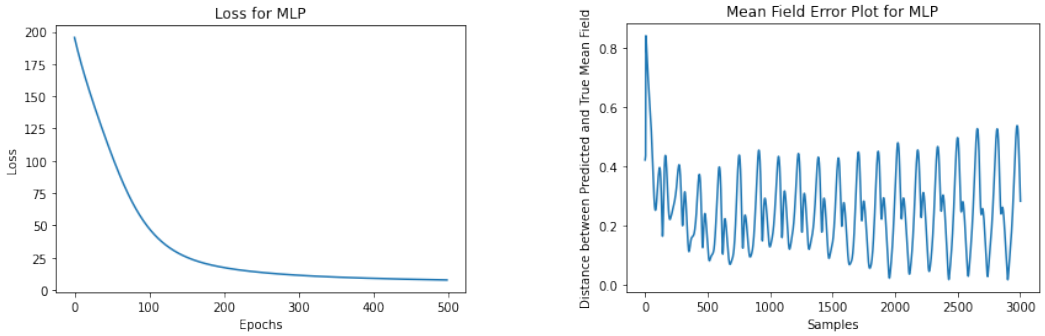


Figure 10: Training loss for model trained on Steady State with same different conditions as test data

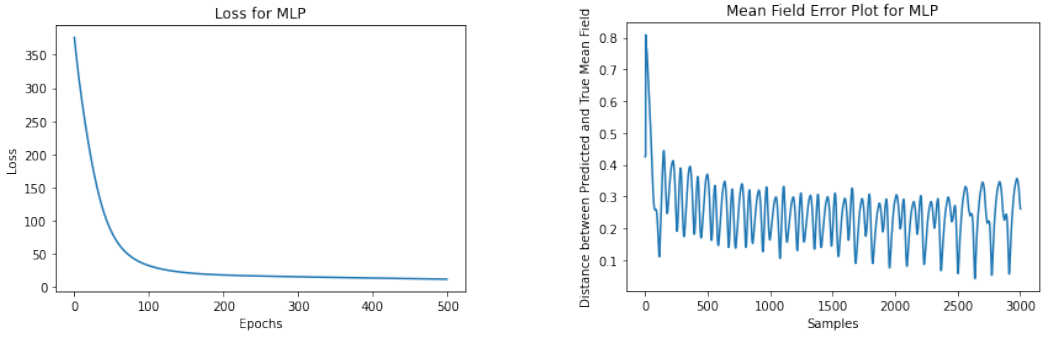


Figure 11: Training loss for model trained on Transient with different initial conditions as test data

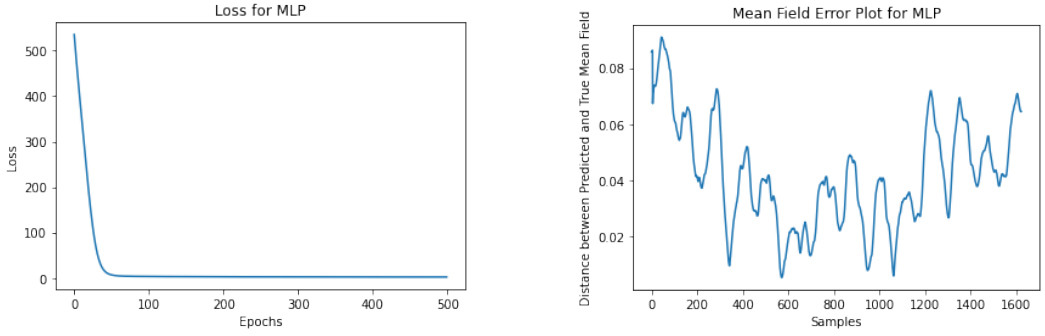


Figure 12: Training loss for model trained on Steady State with same initial conditions as test data

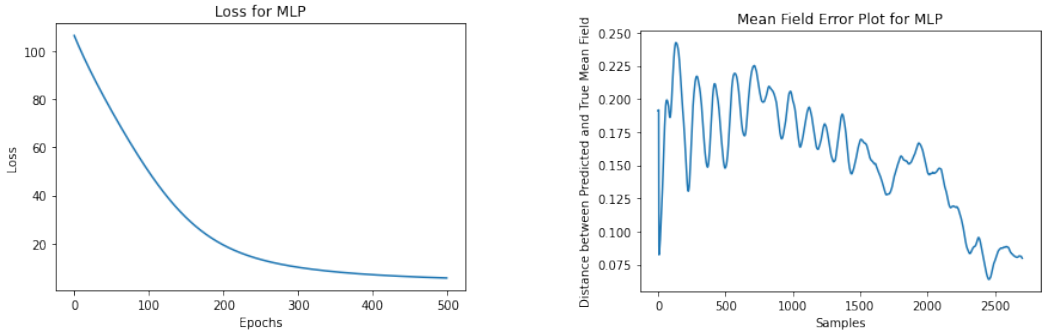


Figure 13: Training loss for model trained on Transient with same initial conditions as test data

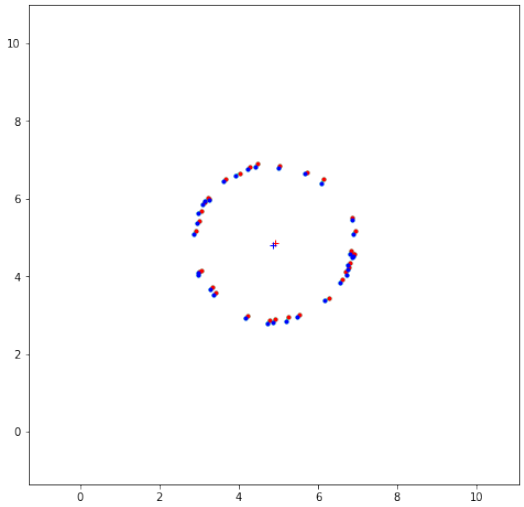


Figure 14: Final Frames of prediction for Steady State with the same conditions (The crosses represent Mean value of the field)

Both the RNN and CNN models performed rather well as well as well while training. They both converged rather quickly while training for all four methodologies mentioned in the Methods section. We also experimented with several different epoch lengths and noticed that lengths more than 50 epochs would consistently produce the same results. The loss plots for the two models in the four cases can be referred from Figures 15, 16, 17 and 18.

The loss plots for all the four cases are as follows:

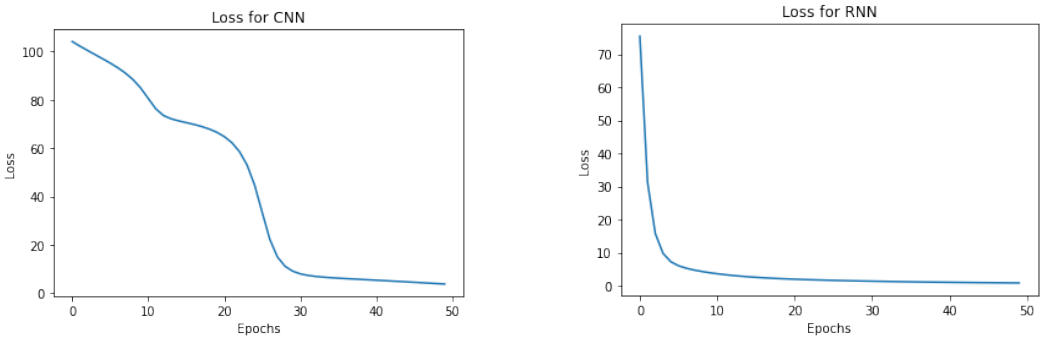


Figure 15: Training loss for models trained on Transient with same initial conditions as test data

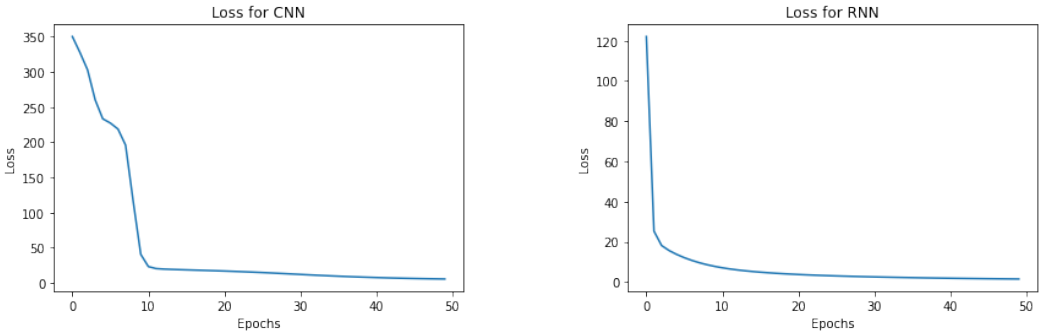


Figure 16: Training loss for models trained on Transient with different initial conditions as test data

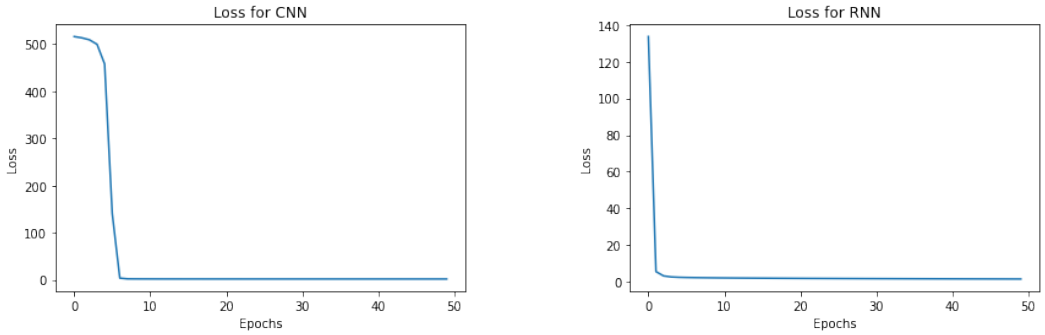


Figure 17: Training loss for models trained on Steady State with same initial conditions as test data

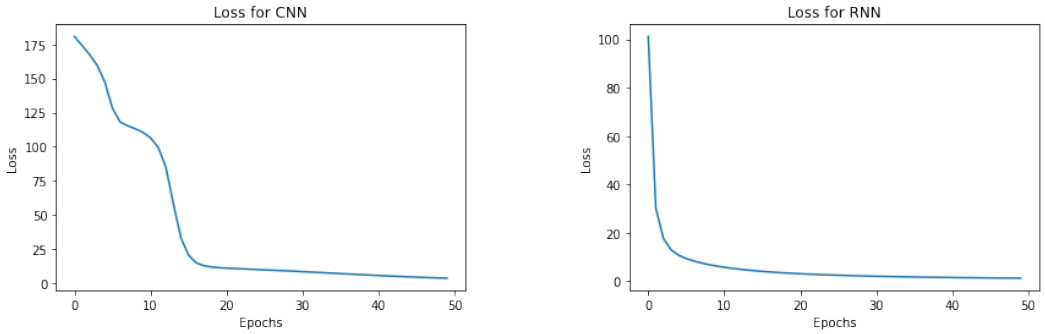


Figure 18: Training loss for models trained on Steady State with different initial conditions as test data

Even though the systems are able to converge to minimal cost rather quickly (50 Epochs), they only evaluate exceptionally well on one methodology - when they were trained on steady-state data with the same initial conditions as the test data. However, on the three other training methodologies results proved to be rather inconsistent as shown by the plots of Mean Field Error below.

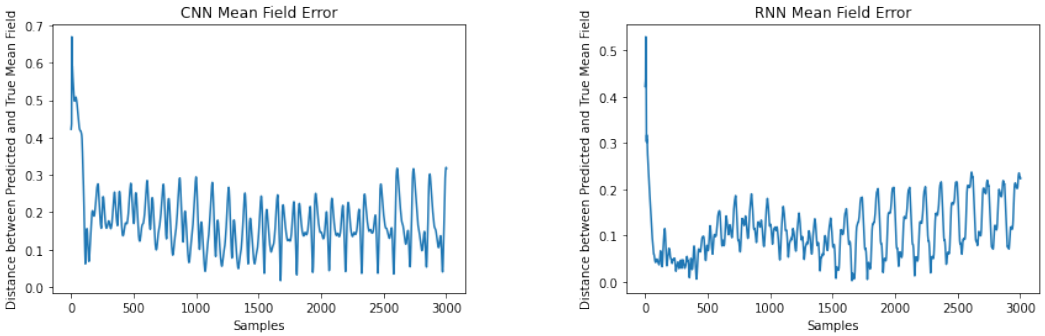


Figure 19: Trained on Transient with same initial conditions as test data

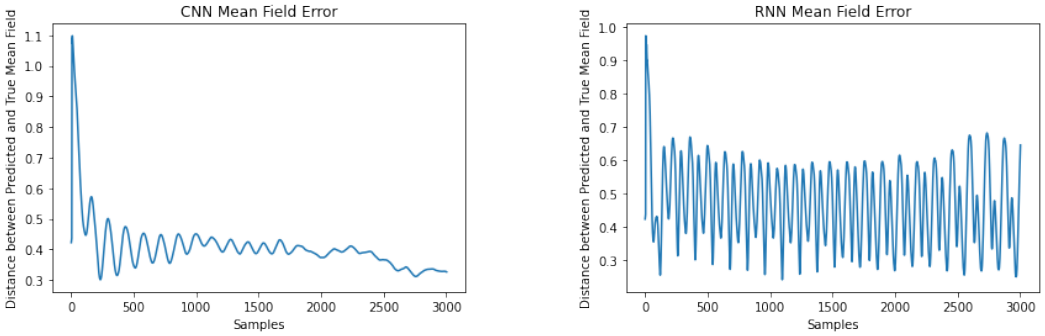


Figure 20: Trained on Transient with different initial conditions as test data

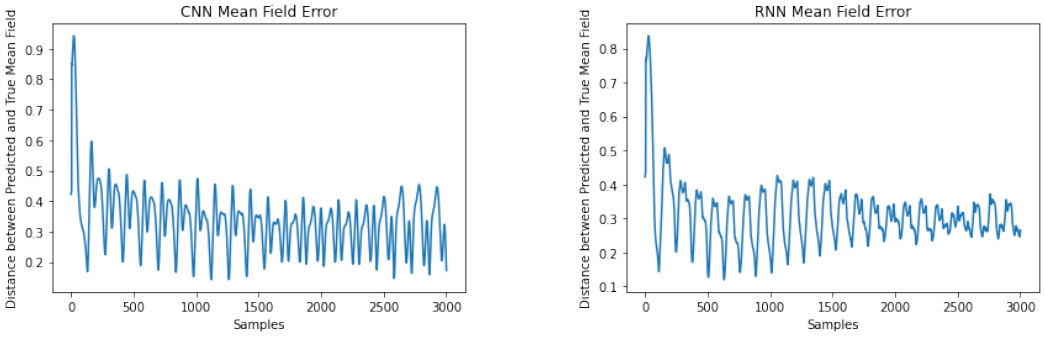


Figure 21: Trained on Steady State with different initial conditions as test data

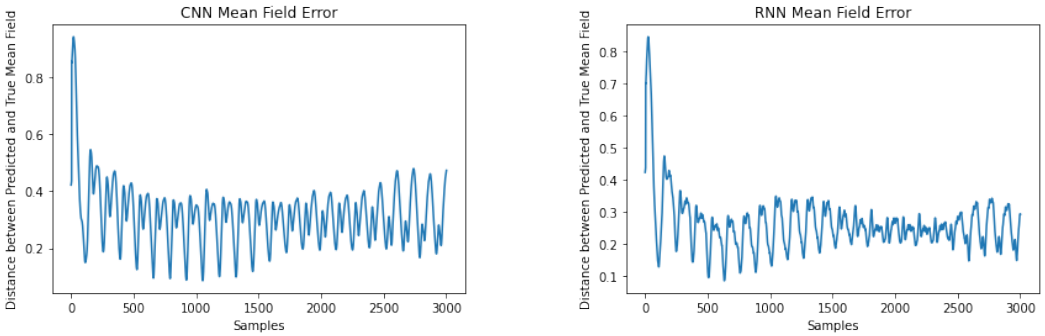


Figure 22: Trained on Steady State with same initial conditions as test data

It is clear that in the above 8 figures is that there is no clear better performing model, this is due to the presence of the transient state in our evaluation data as well. As mentioned earlier it is fully expected the model trained on steady state would perform well, and it does only once the transient state is removed. The mean field error changes rather dramatically as shown in figure 23 versus the same deep learning models that used the same test data but contained the transient states in figure 22.

Once the transient portion of the test data is removed, the average Mean Field Error drops dramatically. This is only the case for the methodology in figure 23. The other plots with truncated test data is consistent to the non-truncated test data which leads us to the conclusion the both RNN and CNN are able to only learn the stable part of the swarm system. The plots for all other truncated tests will be in the appendix. While it may seem that RNN and CNN have very similar performances across the board, observing the animation of the predicted data from each shows that the while the mean field error is similar for both, there seems to be a prominent phase shift in the CNN model. This is clearly seen in the below figure:

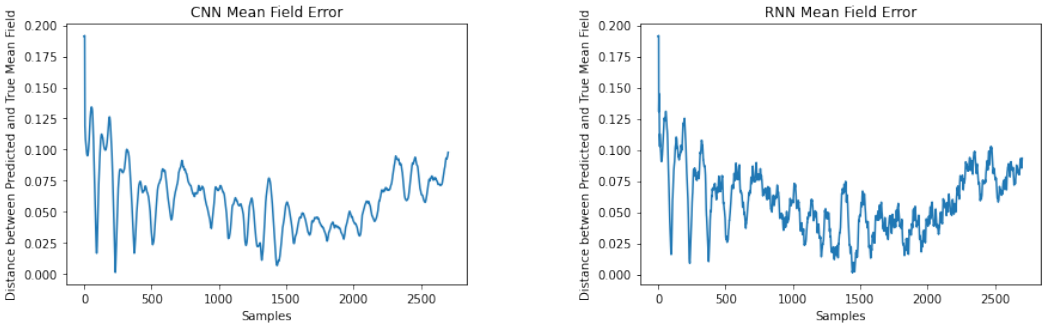
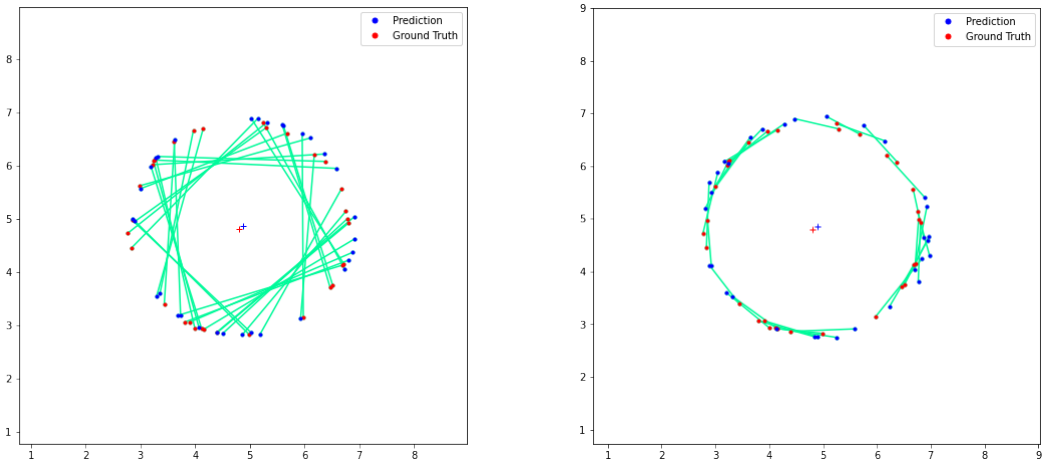


Figure 23: Trained on Steady State with same initial conditions as *truncated* test data



(a) Final frame of CNN Prediction

(b) Final frame of RNN Prediction

Figure 24: Final frames of prediction from methodology of figure 23. [The crosses represent the mean field]

The above figures in this section lead us to the conclusion that the RNN and MLP is slightly better in learning intera-particle interactions and individual particle behaviours as opposed to only learning the steady state of the entire system which seems to be what is happening with the CNN. This led us to conclude that the MLP, RNN and CNN models are well suited to learning steady state models from data with same initial conditions but with the presence of transient states or states from different initial conditions, they do not learn well. Fundamentally, they are not able to generalize well enough. Thus all the three deep-learning based baselines performed very similarly.

Similar to the non-baseline models, the deep learning baselines are not able to identify the hidden state dynamics of swarm system and finds difficulty in capturing the highly non-linear properties of the transient system.

5.3 Neural ODE

The following evaluation concerns the model without inherent noise modeling.

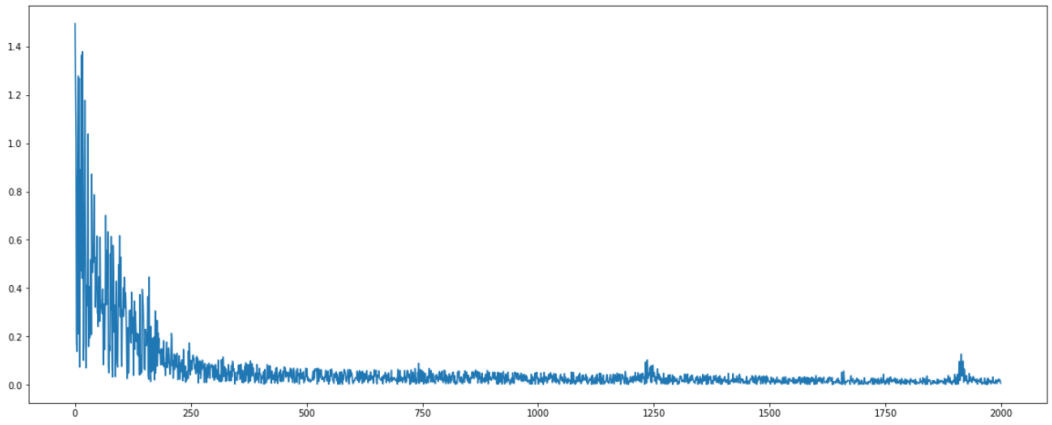
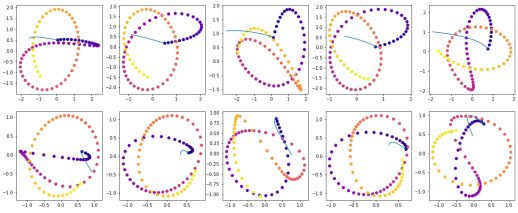


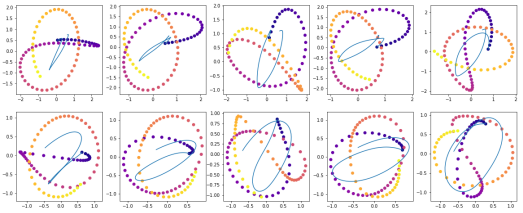
Figure 25: Neural ODE learning curve

Neural ODE in conjunction with physics-informed network was the most promising model we have come up with. The trained network not only shows long term convergence to the correct steady state, but is also robust to different initial conditions. Figure 25 shows the learning curve for 2000 epochs. The model converges quickly through the first 500 epochs. Though the mean squared errors don't seem to decrease much after 500 epochs, it can be visually inspected from the training process that the predicted trajectory is in fact still moving closer towards the true trajectory.

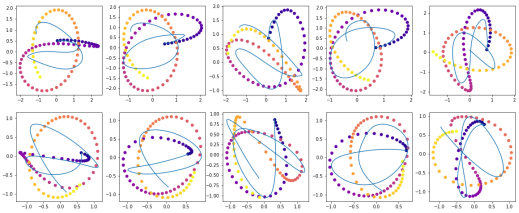
The training process was visualized and shown in figure 28 and 31. The former figure shows 2000 epochs learning of a noiseless swarm system while the latter shows the learning of a noisy swarm. In both cases the model converges. This demonstrates that our training method is robust to noise in dynamics.



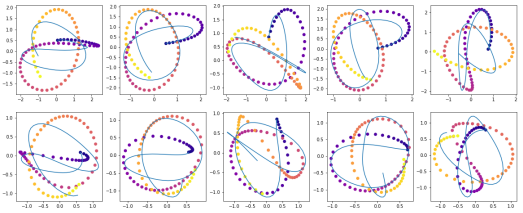
(a) epoch = 0



(b) epoch = 126

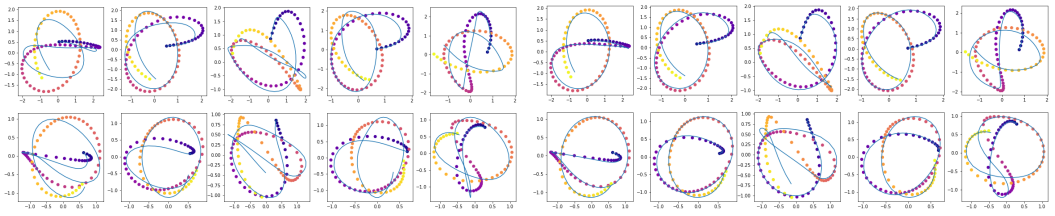


(a) epoch = 250



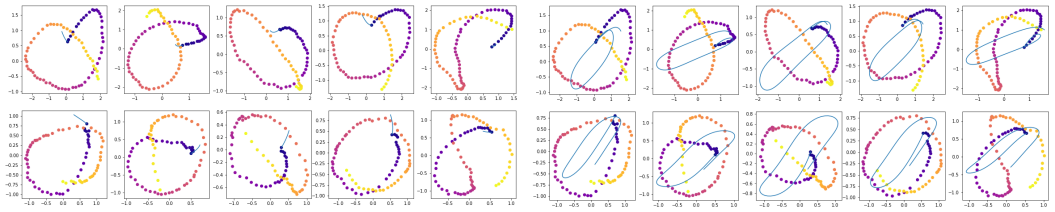
(b) epoch = 500

It was mentioned in the introduction that two criteria are important for evaluating the learnt model for swarm behaviors. The first is the convergence to steady state as time approaches infinity. Figure 32 shows the simulation of the ground truth trajectory and the learnt trajectory for a very large number of steps (100 time steps). It can be seen that the blue lines eventually circle around like the true steady state and it does not have the tendency to escape the steady state. Many similar

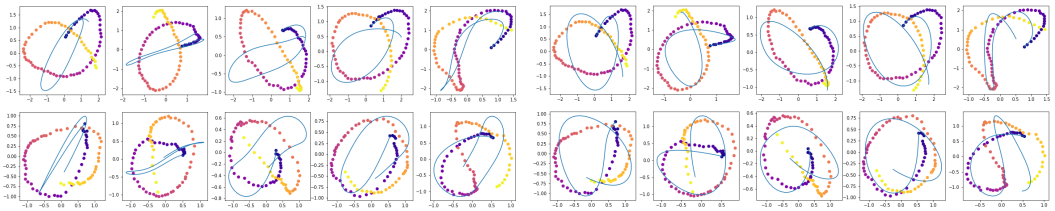


(a) epoch = 1000 (b) epoch = 1998

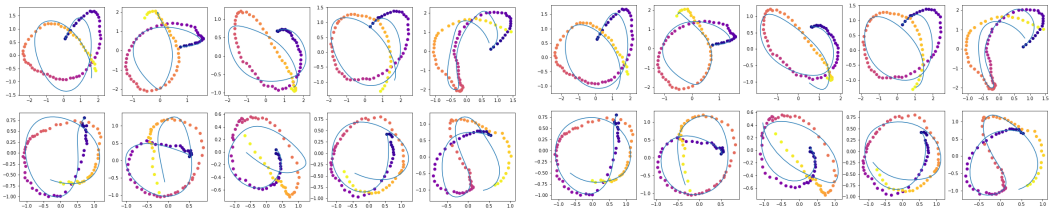
Figure 28: Learning a noiseless swarm from the transient data. For each plot, the 5 graphs in the top row are the positions of each agent, and the 5 graphs in bottom row are the velocities of each agent. Dots are true trajectories, with darker color being earlier in time and lighter color being later in time. The blue lines are the trajectories predicted with the trained model.



(a) epoch = 0 (b) epoch = 130



(a) epoch = 250 (b) epoch = 500



(a) epoch = 1000 (b) epoch = 1990

Figure 31: Learning a noisy ($\mu = 0, \sigma = 1$) swarm from the transient data. For each plot, the 5 graphs in the top row are the positions of each agent, and the 5 graphs in bottom row are the velocities of each agent. Dots are true trajectories, with darker color being earlier in time and lighter color being later in time. The blue lines are the trajectories predicted with the trained model.

experiments were run and all of them remain in steady state. Though global convergence as time approaches infinity needs rigorous mathematical proof to claim true, empirical results are sufficient for preliminary convergence claims for the purpose of this paper.

Convergence to steady state is further supported by figure 35, which are plots of mean field errors for prolonged period of simulation (450 time steps and 4500 data points) for 6 different sets of initial conditions. The periodicity indicates the steady state for the trained model, and the mean field error has an average value of 0.4. This is comparable to some of the baseline model results trained with steady state data. However it's important to note that learning from steady state data is a much easier problem because of the fact that steady state dynamics can

inherently be linearized, and therefore learning the steady state is similar to finding a linear approximation. In contrast the transient state dynamics are vert nonlinear and therefore the learning problem is a much harder one. The fact that Neural ODE is able to achieve similar performance by learning from the transient data alone means our final architecture is a more powerful learning method.

The second criteria of a good model is its robustness to different initial conditions. Figure 35 is simulated with 6 sets of different initial conditions. To have more diversified cases, the simulations selected initial conditions from different locations with respect the center of steady state and different scaling. All simulations with the trained model have shown convergence to steady state and roughly correct trajectory prediction even for the transient regimes. This robustness to different initial conditions is unseen in all previous baseline models even for training with the steady state data. Please refer to the video for animations of the predicted swarm motions.

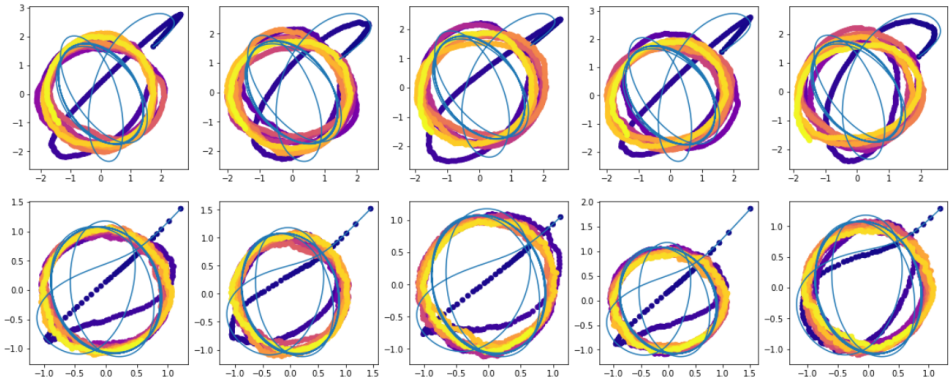


Figure 32: A model trained on transient data of a noisy swarm shows convergence to steady state

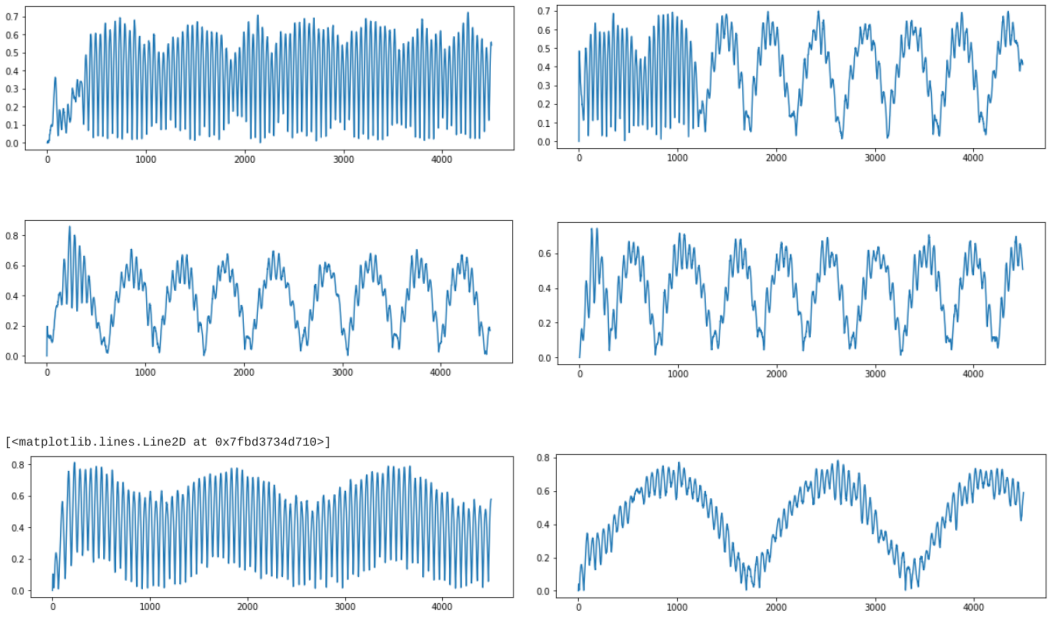


Figure 35: The mean field error of the model prediction on 6 sets of different initial conditions

6 Discussion

This project shed a lot of light on the evolving capabilities of existing neural network models and newer models to learn nonlinear ODE systems. We have experimented with non-deep learning baselines, deep-learning baselines and finally the Neural ODE.

Initially starting out with this project, we aimed to test out various existing deep learning methods to learn from time-series data that was generated by swarm systems. While the non-deep learning baselines and deep learning methods we used were able to quickly learn the stability of the system and evaluated well with systems with the same initial conditions, they were not able to learn very well on transient states and evaluated poorly on test data with different initial conditions.

This corroborates the linearizability of stable equilibrium of dynamical systems, a well-studied subject as a sub-branch of mathematics. It also makes sense that the transient states and swarm with different initial conditions cannot be approximated with these models because the approximately linear function learnt does not represent the full dynamical process.

In general, all our baselines were not robust enough to handle transient states, different initial conditions and the inherent nonlinearity of the swarm systems. This led us on the search for more SOTA methods which led us to Neural ODE which is a rather new deep learning method that provides researchers with the tools to better capabilities to train ODEs within larger models. This, however, needs to be utilized together with well designed dynamics models in order to truly grasp the nonlinear dynamics of a swarm. We had a stroke of luck in finding a working model but in general such a model may be hard to find and sometimes the modeling process might impose assumptions too strong to be transferable to real applications.

It's also not clear how does such a network in conjunction with Neural ODE captures the true dynamical process. In other words, we are not sure why our model gives good results. This is because bifurcations and catastrophes of dynamical systems would mean that the stability and other properties of the system is in fact not unique to the same set of parameters. The fact that our model has hundreds of parameters means bifurcations and catastrophes are more likely to happen. In addition, dynamical systems can exhibit hysteresis, or in other words, the change in stability as one sweeps across the parameters is not the same for both directions. These potential properties of dynamical systems may all be a hindrance to learning a good model.

6.1 Future Work

Since the Neural ODE works as black-box system, it would be interesting to test the model with architectures such as recurrent layers, LSTMs, convolutional layers or even a combination. There are several possibilities with this new development and this method may have a large impact on the study of robot dynamical systems.

An important future area of work include the mathematical proofs for network convergence. There has been work on co-training Lyapunov functions with a neural network to ensure global asymptotic stability, however, such methods inherently impose assumptions too strong for a system, because the construction of Lyapunov functions is hard and therefore more of an art. By claiming a form of Lyapunov function in the very beginning significantly limits the class of systems that can be learnt. Therefore it might be good effort to design ways to construct Lyapunov functions for neural networks, which allows a larger class of systems to be learnt.

Also the work presented in this paper is confined to one specific swarm model from one particular class. We are confident that our proposed learning method should generalize to more swarm models of the same class and maybe to other classes as well but more work is needed. In addition learning natural swarms and approximate such behaviors on robots are of great interest to the swarm robotics community. Most of such work rely on emergent behavior using state machines rather than dynamical systems, or in other words, they rely on logic rather than mathematics. If natural swarms can be

learnt with the proposed method as a dynamical process, a new class of artificial swarm may spawn by relying on dynamical systems.

Lastly, Neural ODE can be studied more in depth about the effect of using different types of ODE solvers. As demonstrated in our paper, the change in step size can lead to a stability change in our swarm system. It is therefore a curious case how does the ODE solver affect the class of models that can be learnt.

References

[1] Craig W. Reynolds. "Flocks, herds, and schools: A distributed behavioral model". In: (Aug. 1987), pp. 25–34. DOI: 10.1145/37401.37406. URL: http://portal.acm.org/citation.cfm?doid=37401.37406.

[2] Csaba Virágh et al. "Flocking algorithm for autonomous flying robots". In: (Oct. 2013). DOI: 10.1088/1748-3182/9/2/025012. arXiv: 1310.3601. URL: http://arxiv.org/abs/1310.3601%20http://dx.doi.org/10.1088/1748-3182/9/2/025012.

[3] Carl Kolon and Ira B. Schwartz. "The Dynamics of Interacting Swarms". In: (Mar. 2018). arXiv: 1803.08817. URL: http://arxiv.org/abs/1803.08817.

[4] S CHEN, S A BILLINGS, and W LUO. "Orthogonal least squares methods and their application to non-linear system identification". In: *International Journal of Control* 50.5 (1989), pp. 1873–1896. DOI: 10.1080/00207178908953472. URL: https://doi.org/10.1080/00207178908953472.

[5] Pileun Kim et al. "Causation Entropy Identifies Sparsity Structure for Parameter Estimation of Dynamic Systems". In: *Journal of Computational and Nonlinear Dynamics* 12.1 (Jan. 2017). ISSN: 15551423. DOI: 10.1115/1.4034126.

[6] Abd AlRahman R. AlMomani, Jie Sun, and Erik Bollt. "How Entropic Regression Beats the Outliers Problem in Nonlinear System Identification". In: *Chaos* 30.1 (May 2019). DOI: 10.1063/1.5133386. arXiv: 1905.08061. URL: http://arxiv.org/abs/1905.08061%20http://dx.doi.org/10.1063/1.5133386.

[7] John M. Maroli, Ümit Özgüner, and Keith Redmill. "Nonlinear System Identification Using Temporal Convolutional Networks: A Silverbox Study". In: 52.29 (Jan. 2019), pp. 186–191. ISSN: 24058963. DOI: 10.1016/j.ifacol.2019.12.642.

[8] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems". In: (Jan. 2018). arXiv: 1801.01236. URL: http://arxiv.org/abs/1801.01236.

[9] Cristina White, Daniela Ushizima, and Charbel Farhat. "Fast Neural Network Predictions from Constrained Aerodynamics Datasets". In: (Jan. 2019). arXiv: 1902.00091. URL: http://arxiv.org/abs/1902.00091.

[10] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations". In: (Nov. 2017). arXiv: 1711.10561. URL: http://arxiv.org/abs/1711.10561.

[11] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations". In: (Nov. 2017). arXiv: 1711.10566. URL: http://arxiv.org/abs/1711.10566.

[12] Jaideep Pathak et al. "Hybrid Forecasting of Chaotic Processes: Using Machine Learning in Conjunction with a Knowledge-Based Model". In: *Chaos* 28.4 (Mar. 2018). DOI: 10.1063/1.5028373. arXiv: 1803.04779. URL: http://arxiv.org/abs/1803.04779%20http://dx.doi.org/10.1063/1.5028373.

[13] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (Mar. 2016), pp. 3932–3937. ISSN: 1091-6490. DOI: 10.1073/pnas.1517384113. URL: http://dx.doi.org/10.1073/pnas.1517384113.

[14] Hassan Ismail Fawaz et al. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* 33.4 (Mar. 2019), pp. 917–963. ISSN: 1573-756X. DOI: 10.1007/s10618-019-00619-1. URL: http://dx.doi.org/10.1007/s10618-019-00619-1.

[15] Tian Qi Chen et al. "Neural Ordinary Differential Equations". In: *CoRR* abs/1806.07366 (2018). arXiv: 1806.07366. URL: http://arxiv.org/abs/1806.07366.

A Appendix 1: Extra Case Studies: CNN & RNN Plots

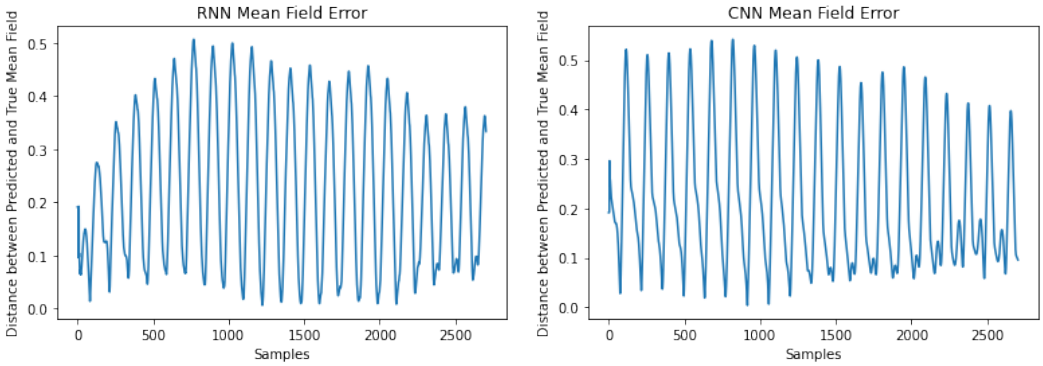


Figure 36: Models trained on Steady State with different initial conditions as *truncated* test data

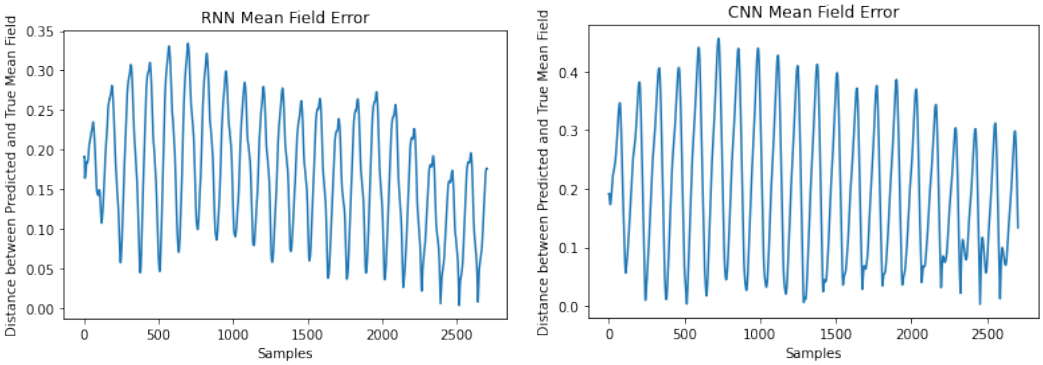


Figure 37: Models trained on Transient State with same initial conditions as *truncated* test data

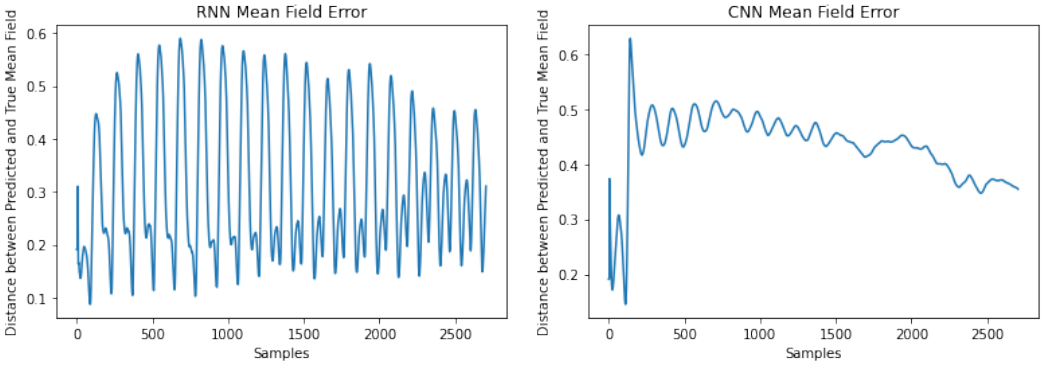


Figure 38: Models trained on Transient State with different initial conditions as *truncated* test data