CIS 620 Project2 Report

Lishuo Pan [panls@seas.upenn.edu], Chunxi Liu [iworld@seas.upenn.edu]
School of Engineering and Applied Science, University of Pennsylvania
Github Link: https://github.com/Paalis/2D_Boids_Meta_Learning

I. Introduction

A. Abstract

In this project, we propose a decentralized controller to learn the multi-agent chaotic system, namely, flocking behavior in a fixed setting and apply the meta-learning to enhance the models' adaptation to a new environment. Our experiments show that meta-learning could enhance the quick adaptation of the established algorithm in the new environment, specifically the training in the new environment setting with pre-trained meta-learner would have a higher convergence rate than the traditional learner. Secondly, the meta-learner could potentially make the algorithm's generalization better, namely, the loss function value in the test set has a lower value.

B. Flocking Behavior Definition

Flocking behavior, which is frequently observed in the natural animal behavior like fish, birds, or herd would emerge a flock moving to a certain direction. As we introduced in the Section I-A, flocking behavior could be categorised as a multi-agent chaotic system [1].

While the underlying dynamics followed by the flocking behavior is not clear. There are works simulate such a behavior by three simple rules: cohesion, alignment and separation. With these three rules, simulated multi-agent system could replicate flocking behavior. The earliest work following this track trace back to 80's [2], credit to people in the computer graphics community.

C. Challenges

One most important character of a chaotic system is that the uncertainty increases exponentially with elapsed time. That is to say, the challenge of such chaotic system is that the system is highly sensitive to the initial condition. This characteristic projects huge difficulty to the prediction model. In our experiment, we found that a well-learned model in the training set suffers generalization in the test set. So our focus of this project would lean more to the following points:

- Design a model which could make good prediction in such multi-agent chaotic system. The details of our baseline model will be introduced in Section IV-C.
- Design a meta-learning algorithm to train the model on a task distribution. Use trained meta-learner as a pre-trained weights initialization to enhance the adaptation of baseline method to a new environment.

The logic of applying meta-learning is that with different initialization chaotic system usually perform dramatically different. This nature of a chaotic system lead us to the conclusion that model learned at the one initialization may suffer a lot from generalization. As a matter of fact, we observe such generalization difficulty in our experiment. The model trained on the training set may not provide an effective prediction nor a good weight initialization. Because traditional machine learning follows assumption that underlying distribution of training and testing set follow the same distribution.

Meta-learning on the other hand is learning to learn. We hope such model could become a effective pre-trained model for our baseline learner to learn the new initial condition. To put this in a simple way, we consider every different initialization in a chaotic system as a domain shift, and we hope meta-learning could help to learn the pattern in domain shift.

***Index Terms*—Meta-Learning, Multi-Agent, Chaotic System.**

II. Relate work

A. Sequence Prediction

Our task fell into the category of sequence prediction. The problem is usually defined as given the input sequence $[x_1, x_2, \cdots, x_n]$ and corresponding label sequence $[y_1, y_2, \cdots, y_n]$, learn the prediction model such that predict y_k given a new input x_k.

This is a well studied area, traditional machine learning approach to such task would be RNN-based methods. LSTM and GRU extend RNN method to fix RNN's defeat in the long-term prediction [3].

B. Boids Flocking

Boids flocking problem is first introduced by Reynolds in 1987 [2]. Following studies showed that the models introduced by Reynolds has chaotic character [1].

III. Dataset

The ground truth data is obtained from a 2D physical simulation for boids, which generates the sequential state information for a given number of agents [1]. The state information includes the positions (x, y) and orientations (dx, dy) of every agent at every time step. 40 simulations are ran in which the trajectories, over 500 time steps, of 8 agents with slightly different initial starting positions from simulation to simulation are collected to be used for training, validating and

[1]https://github.com/jackaperkins/boids

testing our models. Although the initial starting positions are really close for all simulations, the trajectories of the flocking behavior are drastically different.

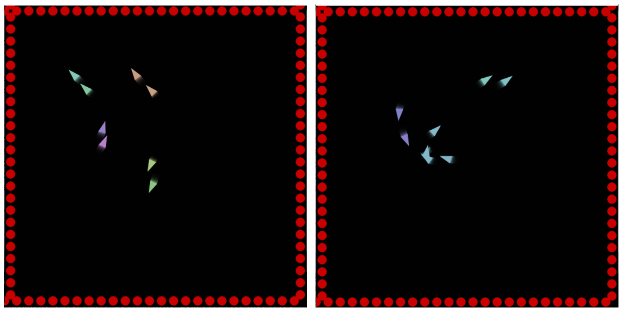

Fig. 1. Beginning positions (left). A simulation in progress (right).

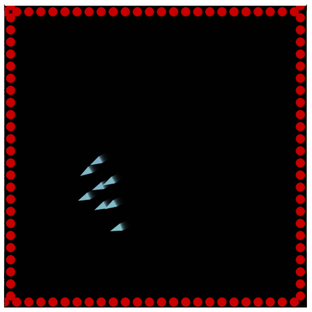

Fig. 2. A flock is formed at the end of each simulation

IV. Baseline Method

A. Problem Formulation

Our baseline model is a general decentralized dynamics for multi-agent system. We notify the whole system as X_t as the state of multi-agent at time t. Agent i state at time t is noted as x_{ti}. Our model $f : \mathbf{R^4} \to \mathbf{R^4}$ learns the state shift ΔX_t given input X_t.

$$\Delta X_t = f(X_t)$$

In our model, given the initial condition X_0. The following prediction state is output of network $f()$ as $\hat{X}_{t+1} = \hat{X}_t + \Delta\hat{X}_t$, where $\hat{X}_0 = X_0$.

B. Decentralized Model

The proposed model is a decentralized model. The input to the network $f()$ is the current agent with its state information. The neighboring agents will be considered when they enter the current agents perception range R_i. In the implementation, we simply concatenate all the agents' information under current agent's state information and filter out those not in the perception range as 0.

Decentralized design provides us with the flexibility to adapt to k multi-agent system where k is chosen as we like.

C. Method Details

For the baseline model, a RNN-based model (with GRU unit and ReLU activation) is used. The model uses a multi-layer gated recurrent unit (GRU), which is a gating mechanism in recurrent neural networks (RNN). For each element of the input sequence, a GRU layer computes the reset gate (r), update gate (u), and new gate (n), along with the hidden state (h) at time t in the following way:

$$r_t = (W_r x_t + U_r h_{t-1} + b_r)$$
$$z_t = (W_z x_t + U_z h_{t-1} + b_z)$$
$$n_t = tanh(W_h x_t + U_h(r_t * h_{t-1}) + b_h)$$
$$h_t = (1 - z_t) * n_t + z_t * h_{t-1}$$

Where W, U, and b are weight matrices and bias.

The input into the model at each time step for each agent is the flattened vector containing the states of current agent and neighboring within its perception range described in Section IV-B. This will be a nk tensor, assuming there are k agents in the system.

The model consists of a GRU unit and 2 fully-connected layers. The input is first passed into a 4-layer-GRU along with a hidden state initialized as 0, where the input of i^{th} ($\mathrm{i} \geq 2$) layer is the hidden layer of the previous layer multiplied by a dropout. The tanh activated output of the GRU unit is passed through a tanh activated fully-connected layer into the final output fully-connected layer.

The model is trained on data collected from single or multiple simulated trajectory. The objective is to minimize the MSE loss between the predicted state shift $\Delta\hat{X}_t$ and ground truth ΔX_t.

$$MSE(\Delta\hat{X}_t, \Delta X_t) = \frac{\sum_{t=1}^{n-1} ||\Delta\hat{X}_t, \Delta X_t||^2}{n-1}$$

V. Meta-Learning method

To handle the difficulty introduced in Section I-C, we utilize meta-learning [4] [5] and treat each trajectory simulation as a different task. With meta-learner learned at the training task distribution, we hope with a different initialization, the model with pre-trained meta-learner as weights initialization could adapt to the trajectory quickly.

The meta-learning framework in our project follows the general meta-learning design [4]. The algorithm is summarized as following:

Algorithm 1 Simplified Model-Agnostic Meta-Learning

```
Require: $p(\mathcal{T})$ : trajectory task distribution
Require: $\alpha, \beta$ : step size hyperparameters
1: randomly initialize $\theta$
2: **for** $\mathcal{T}_i \sim p(\mathcal{T})$
3:     Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
4:     Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
5: **end for**
6:     Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
```

Algorithm: Simplified Model Agnostic Meta-Learning

VI. Experiments

A. Experiments Metric

To quantify the performance of our model in terms of forming flocking behavior. We use position consensus metric to quantify the pair-wise distance with the multi-agent system. position consensus for time t is defined as,

$$\sum_{i \neq j} ||X_{ti} - X_{tj}||^2$$

In additional to position consensus metric, we also inspect the $2D$ static image of generated trajectory to see if the flocking has emerged.

B. Baseline Performance

In this section, we trained the baseline model on the simulated trajectoies from one simulation and then on the simulated trajectories from multiple simulations, and performed interpolation and extrapolation experiments for each trained model.

1) Baseline Part 1– Training on one simulation data: We first trained the baseline model on 500 time stamps of simulated trajectories 5 from the beginning. We trained 1000 epochs and stored every 10th. To ease training, because our data is continuous, we used a sampling rate of 2, by taking every other data points to create the datasets.

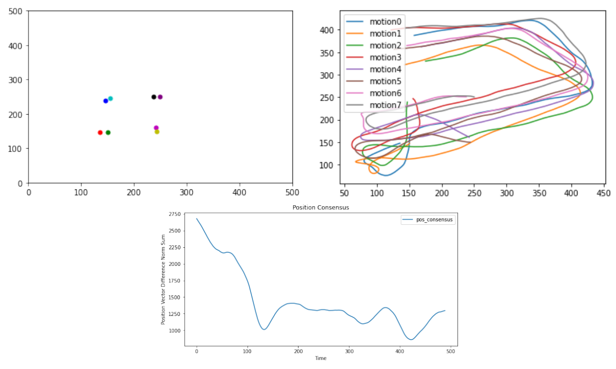

Fig. 3. Simulation 5: Initial agent positions (left). The trajectories for the first 500 steps (right). Position consensus (bottom).

Experiment 1: For interpolation, the model predicts the agents' trajectories from the initial condition of simulated trajectory 5, which is the data we trained on.

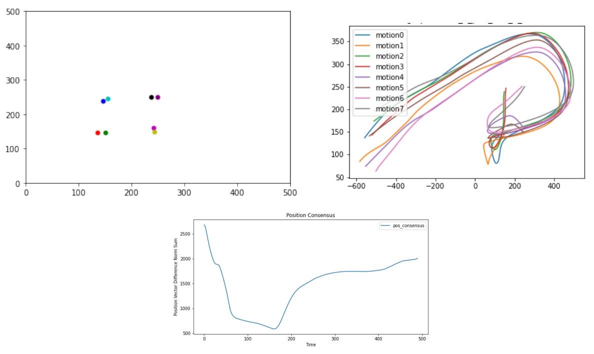

Fig. 4. Experiment 1 Interpolation: Initial agent positions (left). Predicted trajectories of the model trained after 900 epochs (right). The position consensus (bottom).

Experiment 2: For extrapolation, the model predicts agents' trajectories from a set of initial condition that has not been seen in training. Specifically, we used the agent initial condition in simulation 20 at time stamp 100.

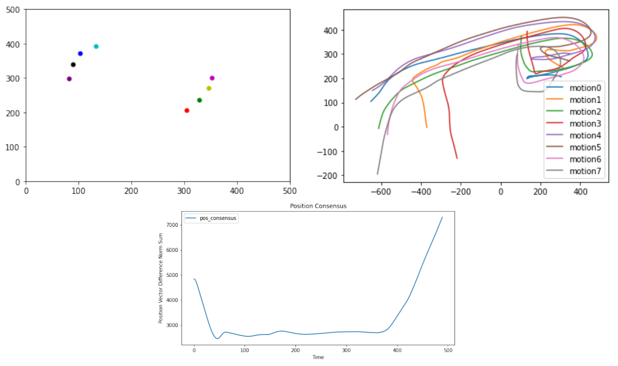

Fig. 5. Experiment 2 Extrapolation: Initial agent positions (left). Predicted trajectories of the model trained after 800 epochs (right). The position consensus (bottom).

We discovered that for both interpolation and extrapolation, the trajectories are pretty much of the same shape "$\supset$" as seen in simulation 5, with the interpolation shape being closer to the actual data shape, which makes sense because the model has only learned one type of trajectory and that's from simulation 5. However, the position consensus of both experiments do not closely match up with that of simulation 5, especially in extrapolation, as it diverges after converging. Also, the absolute positions of the agents in both experiments are quite off as they wander together to the far left.

Additionally, we noticed that training is rather unstable as increasing the number of epochs does not necessarily improve the predictions.

2) Baseline Part 2– Training on multiple simulation data: We then trained our model on a dataset containing multiple simulations, specifically, simulation $0, 1, 2, 3$ and 4.

Experiment 3: For interpolation, this model predicts the agents' trajectories from the initial condition of simulation 0, which is used in training. The initial positions of simulation 0 are very similar to simulation 5.

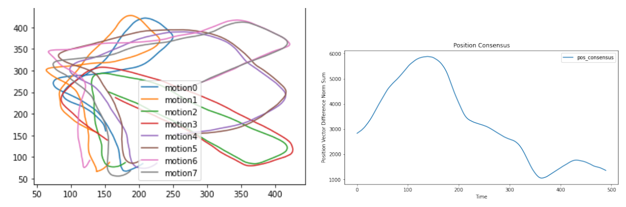
Fig. 6. Simulation 0: The trajectories for the first 500 steps (left). Position consensus (right).

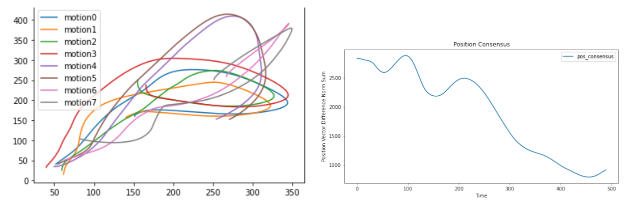
Fig. 7. Experiment 3 Interpolation: Predicted trajectories of the model trained after 100 epochs (left). The position consensus (right).

Experiment 4: For extrapolation, the model predicts the agents' trajectories from a set of initial condition that has not been seen in training. Specifically, we used the agent positions in simulation 20 at time 100, same as in experiment 2.

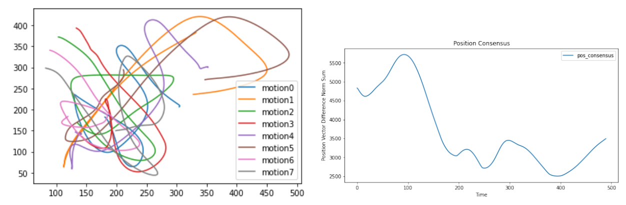
Fig. 8. Experiment 4 Extrapolation: Predicted trajectories of the model trained after 990 epochs (left). The position consensus (right).

From experiments 3 and 4, we found out that the model trajectories no longer follows one single shape, which is expected since it is trained on a variety of trajectories from different simulations. Besides, the model performance seems to improve in both cases as the position consensus value decreases over time, which indicates the emergence of a flock. By comparing experiment 2 with experiment 4, where both are predicting from the same initial positions, it is clear that training on multiple simulation data improves the model performance when coming across a novel initialization.

C. Quick Adaptation of Meta-Learning

In this session, we compare the quick adaptation capability of baseline model and meta-learner model.

1) Meta-learning pre-training compared to baseline pre-training: We train two networks as our pre-trained models, one with baseline algorithm and the other use the meta-learning algorithm. Two models are trained on simulation dataset $0-4$. In meta-learning setting, one thing worth noticing is that one meta-learning update will need the training to go through the whole dataset, while in the baseline training, each trajectory in the dataset will update network once. For a fair comparison, we use the 500 epoch of meta-learner and 100 epoch of baseline learner as our two pre-trained models.

Figure 9 shows pre-trained models' loss value descent,

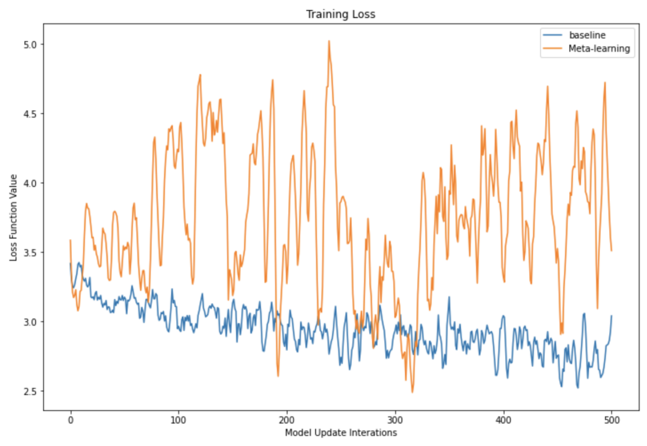
Fig. 9. Loss function value descent for meta-learning and baseline algorithm pretraining

2) Meta-learning pre-training compared to baseline fine-tuning - quick adaptation: To adapt the pre-trained model to new initialization condition, we initialize coefficient in the baseline model with these pre-trained models and fine-tuning on the new simulation trajectory simulation data 5. Figure 10 compares the loss value descent on the these two experiments,

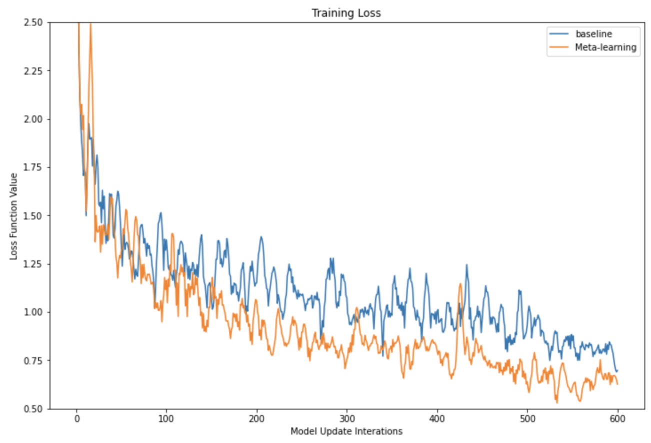
Fig. 10. Loss function value descent for meta-learning and baseline algorithm for the quick adaptation experiment

From figure, we could observe that the loss value descent of meta-learning algorithm is much faster than the baseline method given that the initial loss of meta-learning is higher than baseline.

3) Flocking Behavior Comparison: We also compare the visual and metric of flocking behavior of baseline and mate-learning. The discussion of this session is based on model performance on simulation trajectory 5. The ground truth of trajectory 5 is shown in Figure 3.

The generated trajectory of mate-learning and baseline method at fine-tuning epoch 500 are shown in the following figures:

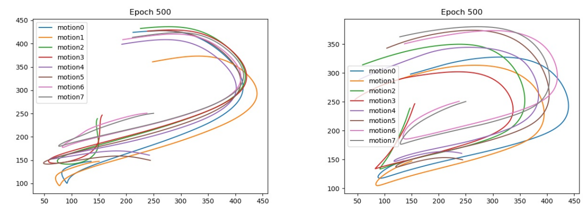

Fig. 11. Generated Trajectory at 500 epoch fine-tuning for meta-learning (left). Generated Trajectory at 500 epoch fine-tuning for baseline (right).

The position consensus of mate-learning and baseline method at fine-tuning epoch 500 are shown in the following figures:

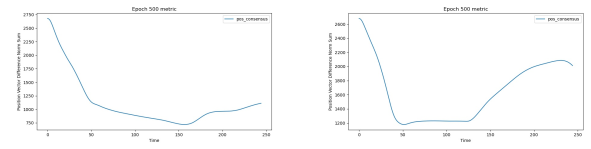

Fig. 12. Position consensus at 500 epoch fine-tuning for meta-learning (left). Position consensus at 500 epoch fine-tuning for baseline (right).

Besides the better performance we found in the meta-learning setting. We also noticed that with meta-learning pre-trained model, compared with baseline pre-trained model. The generated trajectory is more stable across different training epochs. The following pictures illustrate the stability of meta-learner fine-tunning.

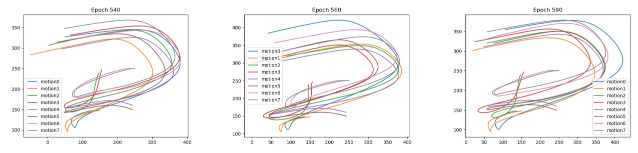

Fig. 13. Meta-learning fine-tuning results at epoch 540, 560, 590

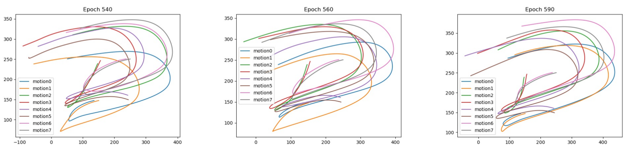

Fig. 14. Baseline fine-tuning results at epoch 540, 560, 590

D. Extrapolation On More Agents

In this session, we directly deploy meta-learning fine-tuned model learned on 8-agent system to 12 agents. The initial states of 12 agents are random. We got following generated trajectory and its position consensus:

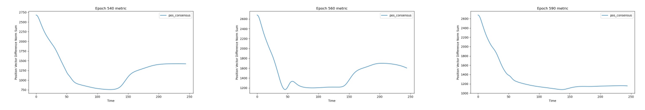

Fig. 15. Meta-learning fine-tuning position consensus at epoch 540, 560, 590

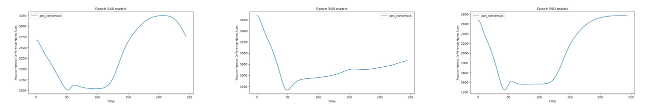

Fig. 16. Baseline fine-tuning position consensus at epoch 540, 560, 590

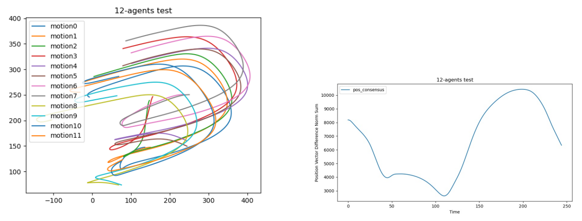

Fig. 17. Model learned on 8-agent system and generate to 12-agent system: generated trajectory and position consensus plot

E. Extrapolation On Time

We also extrapolate the meta-learning fine-tuned model on more time stamps.

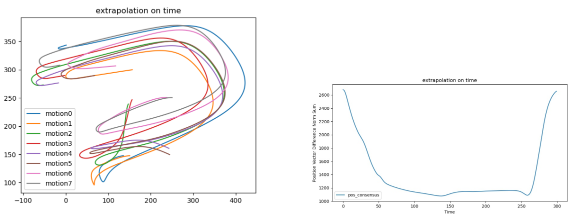

Fig. 18. Model learned on 250 time stamps and generate to 300 system: generated trajectory and position consensus plot

VII. Conclusion

As a conclusion, meta-learning could enhance the quick adaptation of model training. To our surprise, meta-learning fine-tuning make the flocking behavior more stable across different epochs.

References

[1] J. Harvey, K. Merrick, and H. A. Abbass, "Application of chaos measures to a simplified boids flocking model," *Swarm Intelligence*, vol. 9, no. 1, pp. 23–41, 2015.

[2] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[4] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[5] B. Wang, M. Lapata, and I. Titov, "Meta-learning for domain generalization in semantic parsing," *arXiv preprint arXiv:2010.11988*, 2020.