Emergent Pack-Hunting Behavior In Evolved Predator-Prey Interactions

Lukas Mericle,[*] Sandra Viknander,[†] and Dan Qing Li[‡]
Chalmers University of Technology, Gothenburg, Sweden

(Dated: January 10, 2018)

We develop a many-predator/many-prey competition environment that aims to be an effective biological and ecological analogue, to observe and characterize flocking behavior that evolves over time when both predators and prey are learning improved strategies to achieve their goals. We compare the behavior of our model, which uses neural networks to control the agents and steady-state genetic algorithms to train them, to comparable simulations of a modified boids model and active Brownian motion. We find that initially the behavior of our model is similar to that of active Brownian motion, i.e., that the movement is essentially random, but that over time the behavior reflects that seen in the boids model, where agents move according to defined rules that reflect ideal behavior. We observe that the flock of predators in some cases utilize pack-hunting tactics to corral and capture the prey.

I. Background

Collective animal behaviour in the form of flocks is very common in ecological systems. The animals comprising a flock tend to align together or evade the predators, doing so as a flock by means of local interactions.[1] The boids model, created by C. W. Reynolds, is the most significant model to simulate bird swarms, fish schools or other similar animal flocks. In the boids model, the acceleration that each boid is subjected to is governed by three tendencies: cohesion, alignment, and separation, where cohesion is the tendency to be close to the center of the flock, alignment is the tendency to align the velocity with nearby boids, and separation is the tendency to maintain a distance from neighboring boids to avoid collisions.[2] With these three governing factors the boids model is an ideal model to represent the real swarming behavior found in nature.

There is another common model which would represent worst-case flock behaviour: Brownian motion. There are two main forms of Brownian motion: passive and active. The passive Brownian model is simply random movement and effectively models many-body dynamics and mutual collisions. The active Brownian model adds a layer of complexity to the passive form by modelling self-propelled bodies with certain diffusion coefficients that describe the variation of their trajectories and positions.[3] With the active Brownian model, one could expect the prey or predator take energy from their environment and convert it into directed movement with some random fluctuations.[4]

Artificial neural networks are inspired by the structure and properties of biological neural networks that constitute the brains of humans and animals. In particular, feedforward neural networks (FFNNs) have been of considerable interest in recent years as computational ability increases. A FFNN itself is not an optimization method, they are computational structures to which several learning (optimization) methods could be applied, for example genetic algorithms.[2]

The boids model is seen as an ideal model for prey and predators to align with their own flock and avoid or catch members of the other flock. The active Brownian motion model describes random fluctuations of a body's trajectory based on a random distribution with known mean and standard deviation. Our model starts from random initialization of neural networks that control prey and predators, which subsequently evolve and learn through genetic algorithm by competing directly against each other.

II. Motivation

Based on the boids model we investigate a new model wherein prey and predators are involved in an "evolutionary arms race." Both parties will take as inputs the positions and velocities of the nearest agents in each flock and process these inputs with neural networks whose outputs are their respective turning angles for the following timestep. As in real-world evolution, prey benefit when they survive for a longer time, and predators benefit when they can catch prey with less effort.

We seek to study the behavior of the evolving prey and predators, where each flock evolves a separate neural network specification that governs the motion of the individuals, using its own genetic algorithm, thus expanding on the article made by Yen-Wei Chen and others that only sought to determine the coefficients in the boids model itself.[5]

We are interested in comparisons of our model with a modified boids model and the active Brownian motion model. We would like to compare specifically the evolution of the mean-squared displacement (MSD) for both flocks over time, to see if the tendency of our model is to approximate the boids model, active Brownian motion, or both.

III. Experimental setup

Our simulations aim to mirror biological processes wherever appropriate. For our case, this means that a) our agents' sensory abilities are based on how flocking organisms perceive their surroundings, and b) the selection process for control schemes is inspired by

[*] mericle@student.chalmers.se
[†] danvik@student.chalmers.se
[‡] danq@student.chalmers.se

Darwin's theory of natural selection.

There are two flocks of agents: predators and prey. The predator agents' aim is to catch a prey in minimal time, and the prey agents' aim is never to get caught. Capture is defined as any predator agent coming within a certain distance of any prey agent. The agents occupy a square continuous space with periodic boundary conditions such that the topology of the space is toroidal.

Each agent has a constant speed and can only control their turning angle at each timestep of the simulation.

Each agent is aware of a fixed number of other agents for the duration of the simulation. They use the relative positions and velocities of those agents to determine their behavior in the next timestep of the simulation. The agents move according to their control schemes until a single predator agent comes within the capture distance of a single prey agent. The elapsed time is then recorded and used to calculate a fitness score that is passed onto the steady-state genetic algorithm.

A. Codebase

The code was written in MATLAB for release R2017b. Besides the base package, it also uses the Parallel Computing Toolbox. The code can be found in a public GitHub repository.[6]

B. Simulation parameters

We start by defining parameters for our simulations. We limit the number of independent variables and make many of the parameters entirely dependent on those independent variables. This is done in an attempt to minimize the number of arbitrary decisions made when defining a simulation setup.

1. *Constants*

Our constants in all simulations are

$s = 5$	· number of sensed values per agent
$v_1 = 1$	· speed of prey agents

2. *Independent variables*

Our independent variables are

N_1	· number of prey agents
N_2	· number of predator agents
n	· number of "neighbors" of agents
v_2	· speed of predator agents
r_1	· turning radius of prey agents
c	· number of competitions between predators and prey
Δt	· timestep of simulation
t_{max}	· maximum duration of simulation
d_{cap}	· capture distance
P	· population size of genetic algorithm
S_T	· tournament selection parameter
f_{mut}	· mutation frequency
d_{mut}	· mutation distance

From these, we derive a number of variables that are required to define our simulation fully.

3. *Dependent variables*

Our dependent variables are

$n_{11} = \min\{n, N_1 - 1\}$	· number of prey agents that the prey are aware of
$n_{12} = n/2$	· number of predator agents that the prey are aware of
$n_{21} = \lceil 2n \rceil$	· number of prey agents that the predators are aware of
$n_{22} = \min\{n, N_2 - 1\}$	· number of predator agents that the predators are aware of
$r_2 = v_2 r_1$	· turning radius of predator agents
$\Theta_i = \arccos\left(1 - \frac{v_i \Delta t^2}{2r_i^2}\right)$	· max turning angle of agents ($i \in \{1, 2\}$)
$A = 10\pi\left(r_1^2 + r_2^2\right)$	· area of competition field
$L = \sqrt{A}$	· length of each edge of competition field

C. Control behavior

1. *Neural networks*

Each control apparatus consists of a neural network with one hidden layer. The hidden layer is defined to have a number of nodes equal to the average of the nodes in the input and output layers, rounded down to the nearest integer. This is done to minimize the risk of over-learning and also to prevent local minima in the search space from being strong attractors, which could increase the risk of premature convergence of the genetic algorithm.[7, 8]

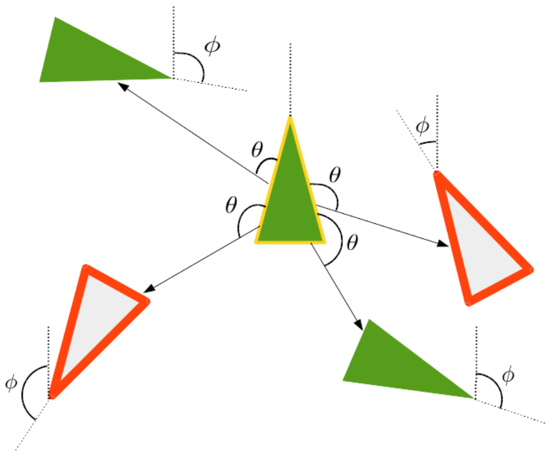

FIG. 1. A visual aid for the environment sensing scheme. The agent that is sensing its environment is in the center, outlined in yellow. The friendly agents that it senses are green, and the foe agents are grey with a red outline. The solid arrows are the displacement vectors, with their length being r. θ and ϕ are defined as the angles of the displacement vector and sensed agent's velocity vector, respectively, in the sensing agent's reference frame.

2. *Environment sensing*

All agents understand another agent's position and velocity in terms of $s = 5$ values. For each "neighbor" agent, they are fed the values $(r/(L/2), \cos\theta, \sin\theta, \cos\phi, \sin\phi)$. If agent a is sensing agent b, then r is the magnitude of the displacement vector between a and b, θ is the angle of the displacement vector (from a to b) relative to a's velocity vector, and ϕ is the angle of b's velocity vector relative to a's velocity vector. The value of the first tuple is normalized such that it falls in the range $[0, 1]$, since each agent can be a maximum of half the field size away in a toroidal topology. Figure 1 provides a visual aid for understanding the parameters collected for each neighbor. In this way, a will understand everything about the position and velocity of b in its own reference frame, thus learning to react only to its local grouping of agents in an invariant way with respect to rotation and translation.

The cosine and sine of the angles are given rather than the angles themselves to normalize the values and eliminate discontinuities in the inputs to the neural networks that serve to control the agents.

Each agent of type i is fed a vector of $s\,(n_{i1} + n_{i2})$ inputs corresponding to the sensed values for the nearest agents of each type. This vector is fed into a neural network with one hidden layer, whose output is a single node that determines the turning angle of the agent when updating its position and velocity.

Each s-tuple is ordered in the vector according to first the type of agent that is being sensed, and second by the proximity to that agent. So all prey tuples are given as the first sn_{i1} dimensions, and all predator tuples are given as the last sn_{i2} dimensions of the input vector. Within each grouping of tuples, the tuples themselves are ordered according to the magnitude of r in ascending order. In this way we enforce the invariance of the sensing apparatus by homogenizing the inputs according to a fixed set of sensible rules.

D. Evolution

1. *Genetic algorithm*

Evolution over time of the control behavior is done with two steady-state genetic algorithms (GAs) — one for the prey controls, and one for the predator controls — that evolve in parallel. A steady-state GA differs from the standard definition in that after all individuals in the population are fully evaluated, we only remove two individuals and generate two new individuals from the remaining population, rather than generating P new individuals from the whole population and dropping all individuals from the previous generation.[9]

In each GA, there is a population of P individuals who each represent a different realization of weights and thresholds for the neural network responsible for controlling an agent. Each individual has a "chromosome" which consists of a list of real numbers, one for each weight or threshold in the neural network. After each generation is complete, the worst two individuals in each population are dropped, and two new individuals are generated from the remaining $P - 2$ individuals via the process described below. Then competitions are run for each of the new neural networks vs. all the neural networks in the other population. This means that for every generation update, only $P^2 - (P-2)^2 = 4(P-1)$ competitions must be performed, rather than the full P^2 as is the case in a normal, generational GA. Because of the long time necessary to perform each simulation, this will result in a massive speedup in obtaining results compared to the normal GA formulation, especially as P gets large.[10]

2. *Fitness evaluation*

We define our problem as a maximization problem. Qualitatively, this means that the prey have a higher fitness when they survive longer on average, and the predators have a higher fitness when they can finish a competition quickly, i.e., catch a prey to end the simulation. This is the same fitness measure used by Gras et al. in their study of a prey-predator simulation, however simplified only to include survival time for the prey and hunting time for the predators.[11]

After c competitions of prey neural network i vs. predator neural network j, the average of the elapsed time over all competitions is stored in a matrix F at the ij position. Once all competitions are finished, the fitness of prey neural network i is defined as $f_{1,i} \equiv \frac{1}{P}\sum_j F_{ij}$, and the fitness of predator neural network j is defined as $f_{2,j} \equiv -\frac{1}{P}\sum_i F_{ij}$.

Once the fitness matrix is completely filled, we sort the matrix and drop the two individuals with the lowest score in each population, then begin with the standard process of a steady-state GA to fill those positions with new individuals derived from the surviving members of the population.

3. *Selection, crossover, and mutation*

Selection occurs on the remaining $P-2$ members of the population by means of tournament selection with the parameter set according to S_T. Then we perform crossover with 100% probability to hedge against premature convergence. Finally, we mutate each new chromosome using creep mutation with a uniform distribution of width $2d_{mut}$ such that the probability of mutation is f_{mut}/m, where m is the number of genes in the chromosome. Typically, f_{mut} is set to 1.[2]

E. Comparison with active Brownian motion

In order to perform an accurate comparison with particles undergoing active Brownian motion, we must simulate the motion with the parameters Ω, D_R, and D_T set such that they coincide most closely with the simulation against which we are comparing.

To obtain the effective values of these parameters for our simulation, we perform backcalculation, using the collected data pertaining to our simulation, on the discretized forms of the equations for simulating active Brownian motion.[4]

First we must find the average turning speed, Ω, which is nothing more than the average turning speed of all agents over the duration of the simulation. In this way, if we record $\Delta\phi \equiv \phi_t - \phi_{t-1}$ for all timesteps for all agents, then

$$\Omega = \frac{\arctan\left(\frac{\langle \sin\Delta\phi\rangle}{\langle\cos\Delta\phi\rangle}\right)}{\Delta t}. \tag{1}$$

We can find the value of D_R by considering the variance of the distribution of $\Delta\phi/\Delta t$ around the effective mean Ω. To do this we employ techniques from directional statistics. First we find the points on the unit circle in the complex plane with the same angle as our observed values of $\Delta\phi$, and we simply find the centroid of those points in the complex plane. Then we calculate the distance R from the origin to that centroid.

To begin, the circular standard deviation is defined as

$$\sigma \equiv \sqrt{-2\log R}. \tag{2}$$

From this we directly calculate D_R by noting that $\sqrt{2D_R/\Delta t}$ serves as the standard deviation for the distribution of angle changes in the discretized model of active Brownian motion. Thus

$$D_R = -\Delta t\, \log\left|\left\langle e^{i\left(\frac{\Delta\phi}{\Delta t}-\Omega\right)}\right\rangle\right| \tag{3}$$

is used as a closest approximation to the distribution of angle changes during a simulation of our model. Note that the expression inside the logarithm will always be of magnitude less than one, so the value of D_R lies in the range $[0,\infty)$.[12]

Finally we must determine the value of D_T. We expect it to be rather low since the agents in our model do not jump in random directions but rather exclusively travel in the direction of their velocity vector. Nonetheless, for the sake of completeness we compute this value.

To obtain the value of D_T we are interested in the variance of our agents' movements after taking into account the "active" contribution to the active Brownian motion.

The variance of the passive portion of the movement in the x and y directions is given by

$$\sigma_x^2 = Var\left[\frac{\Delta x_i}{\Delta t} - v\cos\phi_{i-1}\right], \tag{4}$$

$$\sigma_y^2 = Var\left[\frac{\Delta y_i}{\Delta t} - v\sin\phi_{i-1}\right]. \tag{5}$$

Then we define D_T as

$$D_T = \frac{\Delta t}{2}\left(\frac{\sigma_x^2+\sigma_y^2}{2}\right). \tag{6}$$

Then we simulate a full run of the particles undergoing active Brownian motion using the parameters derived in Equations (1), (3) and (6) and compare its statistics to those generated by our simulation.

F. Comparison with boids model

The boids model we devise for comparison with our algorithm is much like a standard boids model but with a couple specific modifications to be more comparable to our setup.

Firstly, the boid agents do not react to agents within a certain metric distance, but rather to a fixed number of nearest agents in both flocks.

Secondly, as in the article by Yen-Wei Chen et al., the boid agents in our modified boids model experience the expected forces as in a vanilla boids model, but also the predators experience an attractive force toward the prey in its sensory field and the prey experience a repulsive force away from the nearest predators.[5] These forces are constant, i.e., they do not depend on distance.

The fourth force experienced by the predators is defined equivalently to the cohesive force, except that the force is directed toward the nearest prey agents. For the prey, the fourth force is defined equivalently to the separation force, except that the force is directed away from the nearest predator agents.

Thus for each agent, the sum of the forces is of the form

$$F_{tot} = \frac{c_1F_{chsn}+c_2F_{algn}+c_3F_{sprn}+c_4F_{hunt}}{c_1+c_2+c_3+c_4}, \tag{7}$$

where F_{hunt} is the additional force attributable to the agents of the opposite flock, and c_1, c_2, c_3, c_4 scale the weight of each force in the overall effect on a given agent. For our simulations, we want c_4 to be relatively large compared to the other coefficients, to emphasize the urgency with which predators are pursuing food and prey are trying to staying alive.

When collecting the data with which to compare our model against this modified boids model, we initialize the simulation with the exact same conditions as the simulation with which we are comparing, namely that we set the positions and velocities to be exactly the same at initialization.

G. Collecting comparison data

At the end of each generation, once the fitness matrix is completely filled, we determine the best predator and best prey and run a single competition simulation between them. We collect the data during their run and then generate active Brownian motion and boids simulations using those data and the procedures described above. Using these comparison data we can produce analyses of the mean-squared displacement over time as well as visualize the simulation itself.

Other data that we are interested in are the flock polarization and angular momentum data for both our simulation and the boids model. We can compute these values directly as a further means of comparing the behavior between two simulation models.[13]

H. Analysis metric

We seek to understand and quantify the behavior of the different control schemes by means of a dimensionless quantity. One of the most important quantities for measuring diffusion behavior of an array of particles is the mean-squared displacement (MSD), defined as

$$MSD(t) = ||\vec{x}(t) - \vec{x}(0)||^2 . \tag{8}$$

To make this dimensionless, we normalize $MSD(t)$ with the quantity $(vt)^2$, which represents the mean-squared displacement of a particle travelling in a straight line with constant speed v. Then a higher value of this ratio corresponds to more directed, ballistic movement while a lower value corresponds to more diffusive movement. In this way we can characterize the various control schemes according to the evolution of MSD over time.

I. Parallel computing

Since the intent is to evolve our networks over several thousands of generations with several validation runs for each pairing, the efficiency of this setup relies heavily on good utilization of the hardware. To optimize utilization of the hardware, `parfor`, a command deriving from MATLAB's Parallel Computing Toolbox, was used in order to engage multiple CPU threads simultaneously in the algorithm. MATLAB automatically delegated one pairing to each CPU thread. These threads are then run asynchronously without communication between threads to minimize the idle time for each thread.[14, 15]

To even further maximize the uptime on as many threads as possible, the population size of both predators and prey P is set to a specific size. P is defined by Equation (9) where m is the number of available threads and k is the number of simulations per thread. To maximize efficiency you want k to be high so as to allow the variance of the time it takes to run a set of simulations to have a minimal effect on the downtime of all the threads. But a high k also means that each generation takes longer to perform. For our implementation we had 16 available CPU threads. Thus a value of $P = 21$ was used so that each thread runs $k = 5$ simulations during each generation update.

$$P^2 - (P-2)^2 = 4(P-1) = mk \tag{9}$$

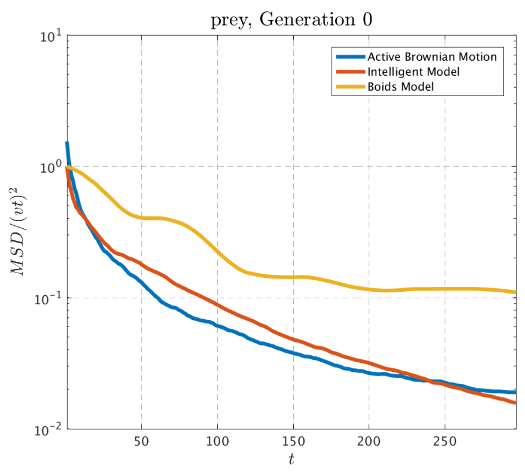

FIG. 2. The displacements of the active Brownian motion, the boids model and intelligent agentsusing prey data in generation 0. The prey display diffusion statistics like the active Brownian particles.

IV. Results

For the simulations we have seen evidence of evolution of both predators and prey. In the first generations we observe that the prey tend to circle around in a small area without flocking, instead diffusing like active Brownian particles as seen in Figure 2. This indicates that they are not using the information of their neighbors' locations and velocities to adjust their trajectories.

The predators do not exhibit a significant difference in the beginning of the simulation but do exhibit a larger displacement than the prey towards the end of the simulation which can be seen in Figure 3. The predators would then best be described as following the active Brownian motion in the beginning of the simulation before transitioning to pursuit behavior like the boids model.

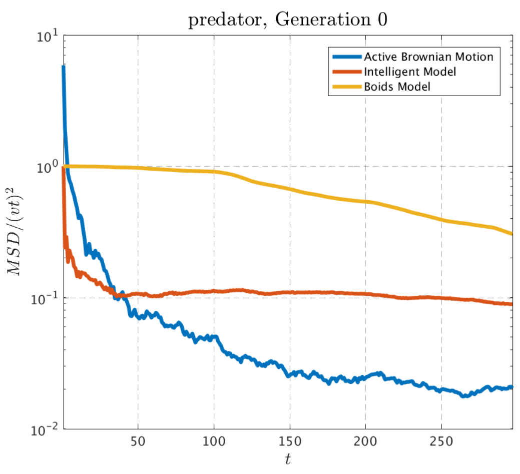

FIG. 3. The displacements of the active Brownian motion, the boids model and intelligent agents using predator data in generation 0. They are initially without direction but quickly transition to pursuit behavior like the boids model.

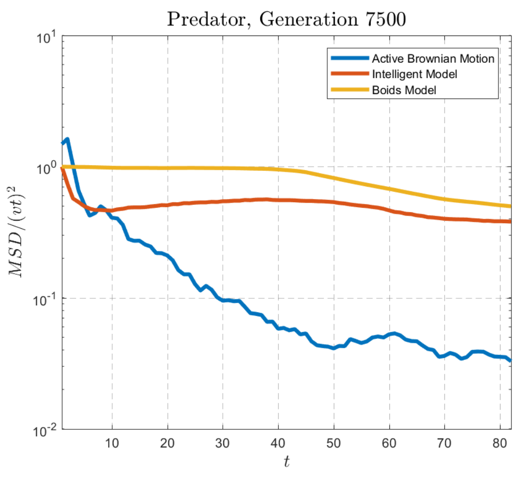

FIG. 5. The displacements of the active Brownian motion, the boids model and intelligent agents using predator data in generation 7500.

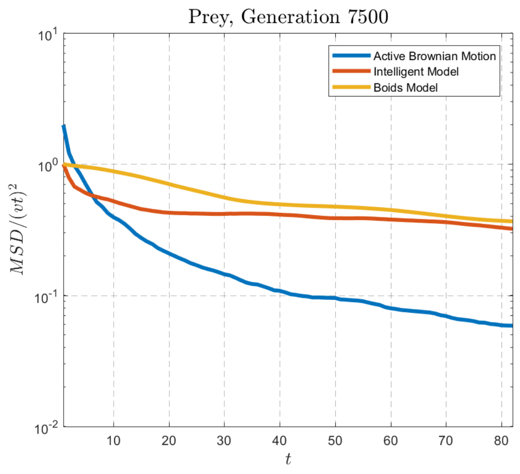

FIG. 4. The displacements of the active Brownian motion, the boids model and intelligent agents using prey data in generation 7500.

For the later generations the prey start out by flocking together and act more like the boids model. This can be seen in Figure 4. The predators also act as in the boids model seen in Figure 5. This indicates some form of flocking behaviour and may also be a sign of pack-hunting behaviour.

V. Discussion

A. Success of algorithm

In our analysis, we assume that active Brownian motion represents a worst-case behavior model for many-predator/many-prey interactions, since random diffusive motion would be trivial to counter by simply moving directly toward the target or away from the hunter. In a similar vein, we assume that the boids model defined above is a perfect behavior model for both predator and prey, since they are responding directly and most effectively to their understanding of their immediate environment for their respective goals.

To that end, we find that the random initialization of neural networks at Generation 0 of the GAs act similar to active Brownian motion and can be said to be moving essentially randomly with no target or "intent." As the generations progress, we find that the models learn some aspects of evasion or pursuit tactics and begin to move in more ballistic paths that signify some intent on the part of the agent, either to catch a fleeing target as a predator, or to flee a pursuing agent as a prey. This mirrors the boids model's setup, whereby the strongest force experienced by the agents is that of either pursuit or evasion, respective of the agent type.

B. Method efficiency

Both the time it takes to run each simulation as well as the large search space for effective neural network configurations are factors that affect the runtime of the algorithm. To combat this we utilized multithreading on the CPU. However we did not utilize perhaps one of the most powerful resources in a computer: the GPU. To further optimize our algorithm the next step would be to move much of the computational load of the simulations to the GPU. To utilize the GPU maximally it would be necessary to have all the validation simulations for each competition with which the CPU threads are working executed at one time. This would however present a need for massive restructuring of our existing codebase, which we felt at the time to be outside of the scope of this project and more suited for a master's thesis project or larger.

C. Future considerations

One thing that would be interesting is to analyze the neural network by studying the "eigenspace" for the hidden neurons by determining the inputs that correspond to a maximal activation at one hidden neuron and a minimal activation at all others. By analyzing the configurations corresponding to these eigenvectors, it would be possible to find neurons that are responsible for behavior in specific scenarios and actions that will be taken after the activation of this neuron. By observing the output and how it is affected by small perturbations at the input the sensitivity of the neural network can be studied.

Another thing that would be of interest is to introduce a tradeoff between intelligence and physical prowess. In such a model both predators and prey would be given a set of evolutionary "points" to spend on their physical abilities as well as their intelligence. The physical abilities of the predators and prey would be their speed and turning angle, while their intelligence would be governed by the number of neighbors they can perceive at one time as well as the size of their neural network's hidden layer. In a setup like this it would be of interest to see if predators and prey choose to go separate paths with respect to brains or brawn. This type of modeling would be computationally heavier than our model. One would also have to decide whether to run each set of evolutionary points over several generations so as to evolve the neural network separately from the evolutionary point allocation, or if you would allow them to evolve simultaneously as in nature. Lastly, it would be interesting to study the fitness values of both prey and predators to observe oscillations between them as well as larger leaps/breakthroughs of either party's ability to accomplish their respective tasks. This would of course be much more interesting to study over a larger set of generations, i.e., more than we have been able to collect for this project. What could be done is also to compare individuals between different epochs to see if later epochs are truly superior or if they just locally optimize to adapt to the behavior of their opponents.

VI. Conclusion

In this project we devised a model of a predator-prey system that lets both the predator and the prey evolve. This exploration has been done by comparing the mean-squared displacement of both predator and prey agents to that of the displacement seen from two other models, namely the boids model and the active Brownian motion model. For the earlier generations both prey and predator showed a resemblance with their displacement to active Brownian motion. The later generations show a closer similarity to the boids model, which would indicate flocking behavior of the prey. This flocking behavior was also observed in simulated videos. Our result here implies that flocking is quite an elementary and essential method for prey to avoid predators since we were able to reproduce this behavior by building a predator-prey system from first principles.

[1] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[2] Mattias Wahde. *Biologically inspired optimization methods: an introduction.* WIT press, 2008.

[3] Julia K Parrish and William M Hamner. *Animal groups in three dimensions: how species aggregate.* Cambridge University Press, 1997.

[4] Giorgio Volpe, Sylvain Gigan, and Giovanni Volpe. Simulation of the active brownian motion of a microswimmer. *American Journal of Physics*, 82(7):659–664, 2014.

[5] Huang X. Nakao Z Chen YW., Kobayashi K. Genetic algorithms for optimization of boids model. *Knowledge-Based Intelligent Information and Engineering Systems. KES 2006. Lecture Notes in Computer Science*, 4252(1):55–62, 2006.

[6] `https://github.com/lukasmericle/ffr120group23`.

[7] Rua-Huan Tsaih and Yat-wah Wan. *A Guide for the Upper Bound on the Number of Continuous-Valued Hidden Nodes of a Feed-Forward Network*, pages 658–667. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[8] Bumghi Choi, Ju-Hong Lee, and Deok-Hwan Kim. Solving local minima problem with large number of hidden nodes on two-layered feed-forward artificial neural networks. *Neurocomputing*, 71(16):3640–3643, 2008.

[9] Gilbert Syswerda. A study of reproduction in generational and steady state genetic algorithms. *Foundations of genetic algorithms*, 2:94–101, 1991.

[10] F. Vavak and T. C. Fogarty. *A comparative study of steady state and generational genetic algorithms for use in nonstationary environments*, pages 297–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[11] R. Gras, A. Golestani, AP Hendry, and ME Cristescu. Speciation without pre-defined fitness functions. *PLOS ONE*, 10(9):e0137838, 2015.

[12] N.I. Fisher. *Statistical Analysis of Circular Data.* Cambridge University Press, 1993.

[13] Iain D Couzin, Jens Krause, Richard James, Graeme D Ruxton, and Nigel R Franks. Collective memory and spatial sorting in animal groups. *Journal of theoretical biology*, 218(1):1–11, 2002.

[14] MathWorks. Parallel computing toolbox^{TM} user's guide, 2017.

[15] G. Yin, C. Z. Xu, and L. Y. Wang. Q-learning algorithms with random truncation bounds and applications to effective parallel computing. *Journal of Optimization Theory and Applications*, 137(2):435–451, May 2008.