SHAPER
Spatially Handled Arm Projectile EvadeR

Ashwin R Bharadwaj
Northeastern University
bharadwaj.ash@northeastern.edu

Aditya Ratan
Northeastern University
jannali.a@northeastern.edu

Andrea Pinto
Northeastern University
pinto.an@northeastern.edu

Hasnain Sikora
Northeastern University
sikora.h@northeastern.edu

***Abstract*—Over the last decade, there has been an accrued improvement in computation capabilities. Organizations all over the world have subsequently used this prowess to develop and harness the power of gradient-based neural networks. This has led to the successful implementation of extensive cloud services and provisioning GPU-intensive ML pipelines. This paper instills the question of exploiting resources for a chance to develop neural networks that allow training without an explicit gradient approach through genetic algorithms and process-based parallelism.[5] We have used this project as a medium to explore genetic algorithms to train neural networks. Albeit a known approach, we have proposed dynamic mechanisms that push the agent to learn better and thus reach its goal in a computationally efficient manner. Our demonstration aims to train our agent which is essentially a robotic arm in a randomised environment. The environment comprises a randomly positioned cannon that shoots polygons at a wall deemed its target. The agent's goal is to prevent the polygon from hitting the cannon's target, which it can do by either deflecting the polygon away, swatting it back at the cannon, or navigating it to the agent's goal box. The agent receives scores that are dependant on a milestone-based score-function optimization which is treated as a fitness function that the genetic algorithm uses to generate the next set of population.**

1. Introduction

The realm of robotics continues to expand its horizons, venturing into intricate domains of collaborative problem-solving and dexterity. Within this landscape, our project delves into the simulation of robotic arms tasked with manipulating polygons shot randomly in a 2D space to the goal state. This pursuit involves adaptive learning, coordination, and the deft manipulation of joint angles.

Using gradient-based neural networks and genetic algorithms for training models is not a new method. There have been successful implementations of these approaches. But they heavily rely on compute-capable resources which more than often are expensive.[8] In this endeavor, the agent is confronted with the dynamic task of navigating through various polygon shapes and orientations, contending not only with the complexities intrinsic to these shapes but also with the gravitational force acting on the polygon. The agent wields control over the joint angles of the robot arm, employing these mechanisms to grasp and maneuver the polygon toward the intended area. Our approach capitalizes on the principles of evolutionary computation, employing GenAI, to refine the agents' strategies iteratively. Through a process of generating and evaluating offspring, we aim to cultivate a cohort of agents adept at collectively manipulating any arbitrary polygon toward its goal. The aim is to design an algorithm that doesn't involve convoluted mathematics and computation thereby eliminating the need for high-performance GPU systems and yet not compromising on the performance of the agent.
Our report delineates the methodology employed, tracking the movements of the agent, the evolution of their strategies, and the adaptation of their approaches toward achieving the project's objectives. Additionally, it discusses the challenges encountered, insights gained, and the potential implications of this collaborative robotic manipulation in broader robotics applications.

1.1. Novelty

We seek to enhance the genetic algorithm's performance by integrating a variable fitness function. This function guides the creation of subsequent populations, allowing the algorithm to learn the ultimate goal while optimizing sub-problems along the way. Additionally, our approach involves dynamically adjusting scores during agent training, gradually transitioning to an improved function upon reaching saturation points. Notably, we aim to circumvent the computational complexities inherent in backpropagation, relying solely on outputs from a feedforward network.

Our simulation introduces novelty in its applicability to fields reliant on robotic deflections or high-precision catching mechanisms. By meticulously selecting inputs and network layers, coupled with developing an environment from scratch, we've crafted a computationally efficient simulation. Importantly, this simulation can be trained on any CPU with multiple cores, ensuring broader accessibility and scalability.

2. Related Works

In [1], the authors present us with a deeper insight into the realm of neural networks and their optimizations using genetic algorithm with a focus on three integral components - the topology, the weights, and the features. The authors delineate weight optimization which involves fine-tuning the numerical importance that the connections of the neurons have, essentially buttressing the network's ability to learn. Topology optimization allows for a computationally efficient network architecture ensuring overall functionality and the optimised feature selection enables the model to only take into account essentially correlated factors for the model to learn from. Genetic algorithm and its crucial focus on population generation, mutation, and crossover probabilities are identified as fundamental determinants that impact the extent to how optimized the networks can become. The paper expounds upon leveraging GA to apply the aforementioned optimizations on support vector machines (SVM), functional link networks, fuzzy neural networks, and extreme learning machine neural networks (ELM).

This paper engenders the question of the prowess of Genetic Algorithm in optimizing a neural network by eliminating the need for weight optimization during back-propagation but instead, going through a series of networks with randomized weights and an integral fitness function to pool and selecting the best-performing networks that can be used on a specified application domain.

In [7], the author expatiates genetic algorithm and mechanisms to optimize neural networks during back-propagation. The paper introduces the classical mutation and back-propagation based mutation which essentially is a pivotal operator in improving the performance of the neural networks in question. the weights of the network are decoded from the chromosome and integrated into the network. Depending on the tasks at hand there are a multitude of cycles of back-propagation are executed until a fit set of weight population is retrieved and re-coded into the chromosome bits.However, despite this process being efficacious, this method is substantially computational and is fairly complex since it is dependent on the weights, the back-propagation and the consistent checks with the fitness functions. The genetic algorithm configuration exhibited exceptional performance and procured a 100% accuracy in diagnosing a singular object of the author's data-set outperforming classical learning methods and also in reduced time frames. However, the genetic algorithm's effectiveness diminished as the number of learnable parameters increased. Our project aims to find an alternative to genetic algorithm's computationally expensive methods and thus create an environment which boasts balance between a set of concrete learnable parameters, feature and complexity of calculations within the network.

In [9], the authors introduce the implementation of evolutionary algorithms to address inverse kinematics for robots operating with n degrees of freedom. The essential focus lies on manipulating the movements through a sequence of some operations performed at designated joints P_1 to P_n withing a clockwise Cartesian coordinate system. The end effecotrs' positions at the operational joints have functions reliant on the n-angles of joint rotations. The fundamental aim was to retrieve a vector of N angles that precisely characterise the desired operational points. To achieve this, the authors used genetic algorithm that various criterion as objective functions during optimization - the first one achieving designated operation points with the suitable precision and the other which served to optimize time, rotations and energy efficiencies. This paper served as an inspiration to integrate genetic algorithm and simulate an arm with n degrees of freedom and select a fit population based on the angles generated by the robotic arm.

3. Methodology

This section is broken down into the intricate components of the agent's spatial properties, the network architecture as well and its environment. The environment and its physics simulation were created using Pymunk and Pygame.[6] We wished to hand-craft an environment from scratch and the aforementioned packages offered a well-documented environment for physics-based simulations and game development. Pymunk served as a lightweight physics engine that enabled the integration of realistic physics into our simulation. It facilitated the integration of gravity, collision detection, and joint/motor constraints, allowing for the creation of dynamic robotic arms and polygons with randomized trajectories. Pygame allowed us a medium to handle graphical updates and to formulate a demonstrative simulation.

A deeper insight into the setup of our environment is delineated in the following section.

3.1. Environment

In this dynamic environment, various components interact to create a challenging scenario for our agent to learn: a Cannon, an Arm with three Degrees of Freedom, a Target Wall, Safe Walls, and a Goal State. The Cannon, positioned along the Y-axis of the Cartesian plane, serves the singular purpose of propelling a polygon towards the Target Wall. The simulation adheres to the laws of physics, incorporating a gravitational force of 10m/s², resulting in a parabolic trajectory for the launched projectile.

At the core of this setup is the Arm, a sophisticated Planar Robotic Manipulator. It consists of a static base, two articulate links designed to shield the polygon from impacting the Target Wall, and an end effector. This end effector's role comes into play once the Arm has mastered the skill of grasping and maneuvering the polygon toward the Goal State. Every 100 generations, the scoring function updates, introducing novel features for the agent to learn and incorporate into its strategies.

The Target Wall, situated opposite the Cannon, acts as the primary barrier that the Arm diligently safeguards the polygon against. Alongside, various Safe Walls exist within the space, allowing the polygon to make contact without adverse consequences.

The Goal State represents an open box strategically placed within the enclosed space formed by the walls. It serves as the ultimate objective, providing the opportunity for the agent to skillfully deposit the polygon inside it, as depicted in Figure 1.

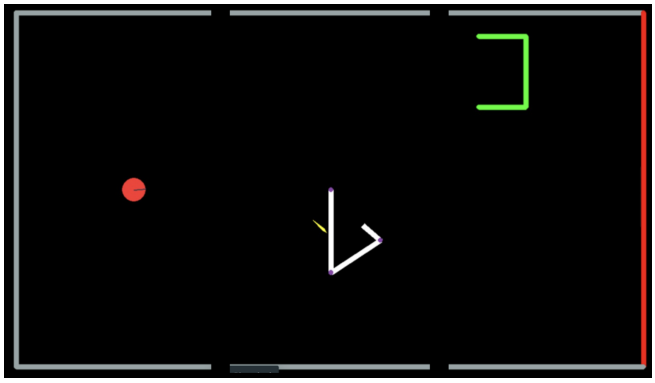

Figure 1. Environment depicting the cannon, polygon, arm, goal box, safe walls and target wall

3.2. Network and Activation Functions

The agent independently utilizes a feedforward neural network [3] comprising three layers: an input layer, a hidden layer, and an output layer. The input layer comprises 28 neurons, each representing distinct spatial and kinematic properties of the environment, including the polygon and the arm. Kinematic features, such as the polygon's position and velocity, in conjunction with the arm joint's angular velocity and acceleration, empower the agent to adjust the motion rates of its joints based on the spatial data of the polygon at any given moment. The hidden layer is composed of 100 neurons, and the output layer consists of 3 neurons responsible for dictating the rotational motion of the arm's joint constraints. Both the hidden and output layers employ a linear activation function expressed as $f(x) = x$. The network's structure is depicted in Figure 2.

Throughout the training process, the network does not directly engage in back-propagation [3]. Instead, a preliminary set of n agents is initialized, each equipped with its unique accessible network. These agents undergo training for a total of 7,200 frames. Each agent generates rotational joint rates for every frame, and a scoring function determines the agent's overall score based on the collision outcomes with the polygon. At the conclusion of their training episodes, each of the n agents accumulates a score, which is then utilized by the genetic algorithm to select the subsequent population set. This evolutionary algorithm

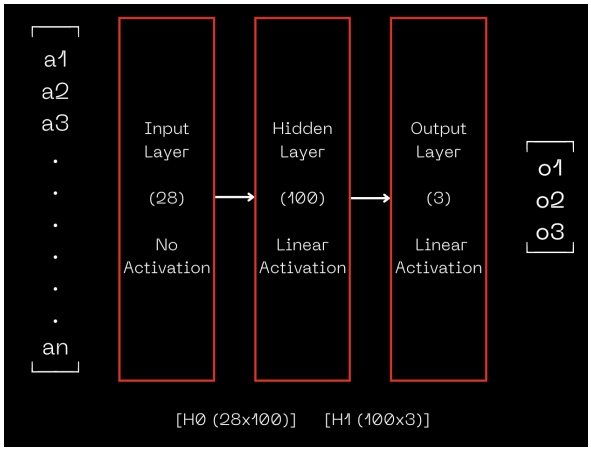

Figure 2. Neural network architecture comprising of 28 input neurons and 3 output neurons.

determines the next group of agents based on those that have achieved the highest scores among the initial population.

This approach obviates the necessity for direct back-propagation and weight updates. Over successive generations, surviving agents are those that excel in maximizing the scoring function. The following sections expound upon the scoring function's nature and the genetic algorithm's [4] operation.

3.3. Genetic Algorithm

This is the crux of the training algorithm. Each agent gets around 7200 frames to interact with the environment for every generation. For each frame in the simulation, the agents receive the kinematic properties of the objects in the environment as input. For each frame, the agent determines the rotation rates of the arms of the bot that it believes will give it the highest reward. At the end of the generation, every agent in the population will have interacted with the environment for 2 minutes.

The agents are then sorted based on the scores they have received over this time. Those agents that have performed better than a certain fraction of the population are considered as parents and the rest are discarded. To maintain the population at a constant level we need to replace the discarded agents. To do so, two random agents are chosen at a time, and using crossover a new agent is created. The new agent is mutated slightly and then added to the population. We do the above process till the population reaches the original level.

We continue the above process till we see that the average score over a few generations has plateaued. At this point, we save the weights of the population and change the scoring function to train the model to reach the next milestone. We continue to do so till all milestones have been reached.

In addition, the amount of mutation [4] applied to the agents that are created slowly decreases as the generation increases. But, if the accuracy stagnates at a lower point the

mutation rate is increased. This acts as a system that can detect and overcome local minima.

3.3.1. Crossover. The process of how crossover works on the constructed genome is delineated by Figure 3. The model was trained using multiple different types of crossover [4] functions.

- Uniform crossover In uniform crossover, each gene(weight in the network) is chosen from either parent with equal probability. This results in offspring which inherit equal genetic information from both parents on average.
- Single point crossover In single-point crossover, a random point is selected on both parent chromosomes(vectorized matrix), and all the genes(weights) to the right of that point are taken from parent A, and the left side is taken from parent B.

The learning rate of the agents was the highest when we used uniform crossover.

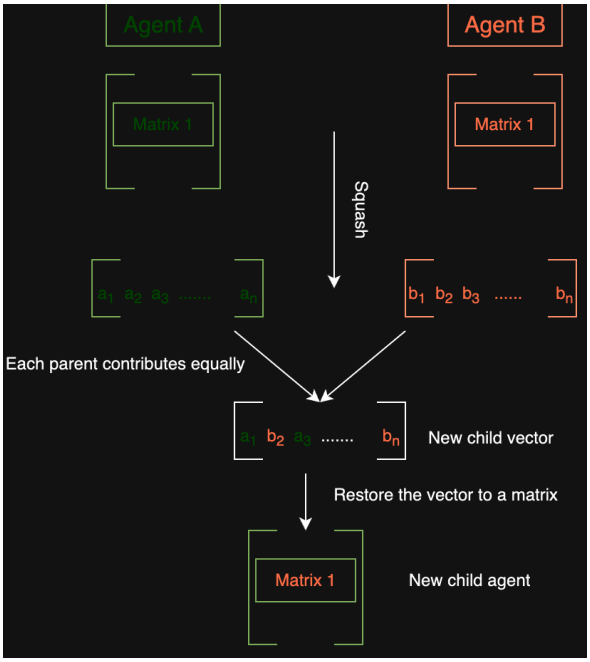

Figure 3. Crossover process

3.3.2. Mutation. The training algorithm mutates every child once it has been created. The mutation rate is slowly reduced over generations but is allowed to increase if the score stagnates. The mutation process is exposited in Figure 4. The model employs two types of mutation:

- Swap mutation. Based on the mutation rate, a fraction of the genes(weights) are allowed to be swapped with each other.
- Uniform mutation. Each weight in the network is slightly altered by adding or subtracting a small random value. The maximum random value is determined by the mutation rate. Once this is done the entire network is normalized between -1 and 1.

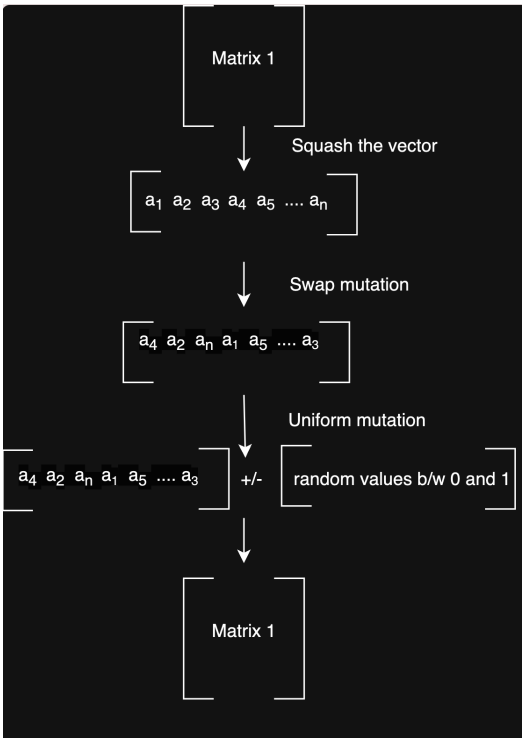

Figure 4. Mutation Algorithm

3.4. Scoring Function

Our strategy for enhancing the agent's performance revolves around a milestone-based approach. The underlying premise is to break down the overarching objective into smaller sub-goals. The rationale is grounded in the idea that by optimizing the agent's performance on these sub-goals and their subsequent functions, followed by a transition to slightly upgraded variations upon saturation, the agent will acquire better learning capabilities for the final goal.

Each milestone essentially embodies a variant of the scoring function utilized by our genetic algorithm to select a fit population. The ultimate aim for our agent is to proficiently deflect a polygon—either causing it to collide with the cannon itself or maneuvering it into the agent's goal box. The essence of our sub-scoring functions lies in determining when and to what extent to reward the agent. [2] There are four distinct scoring functions in our approach as described below.

1) **Contact Priority Scoring Function:** Rewards the agent with +10 when it makes contact with the arm. Throughout the training process, the

arm learned to rotate akin to a propeller. After rigorous training, it tactically acquired the skill to manipulate the polygon using its joints. This scoring function is optimized until the agent's performance stabilizes after which the following scoring function comes into effect.

2) **Safe Wall Priority Scoring Function:** Grants a +50 reward when the agent prevents the polygon from reaching the target. With each successive generation, the agent progressively learned the optimal movements necessary to intercept the polygon along its parabolic trajectory post-cannon launch.
3) **Goal-Box Priority Scoring Function:** Awards +300 when the agent successfully places the polygon into the goal state. While the agent can accomplish this through training, it also learned to discern situations where the cannon's trajectory would naturally align with the goal, thereby abstaining from unnecessary interference.

In addition, if the agent hits the target wall, it gets a penalty of -250. But if the agent deflects the polygon back to the cannon, it is rewarded hugely with $+250$.
After each generation, if the performance average plateaus or fails to improve by a certain percentage, we augment the value of η twofold. This adjustment is aimed at facilitating enhanced learning and mitigating the risk of the agent getting trapped in local minima.

4. Results

In this section, we present the results of our demonstration. As the above sections mention, the scoring function doubles as the measure of our performance as well as the fitness function that the genetic algorithm takes into account to generate a fitter population for the following generation. The population set of agents was initialized and trained for an accumulation of 700 generations with three varied scoring functions.

First, we delineate the accretion of the maximum score collected by the agents for the first 300 generations using the Contact Priority Scoring Function.

Figure 5 depicts the impact of the initial scoring function across the initial 300 generations of training. Notably, there is a consistent upward trend in the maximum achievable score by the agent for each random initialization. To optimize performance, we have implemented a mechanism that doubles the mutation rate parameter η if the average reward stagnates over a 100-generation window, failing to increase by at least 10%.

This phenomenon is observed early in the training process, typically within the first 100 generations. Subsequently, there is a marked and progressive increase in the agent's score accumulation, leading to a substantial improvement before reaching a plateau. Ultimately, the agent's performance stabilizes, with the average score leveling off at around 250,000. At the end of this 300-generation window, the agent has learned to essentially avoid the polygon by swatting it by the arm. It is now time to subject it to a different set of rewards such that it does not just touch the polygon but is also able to deflect it to the safe walls.

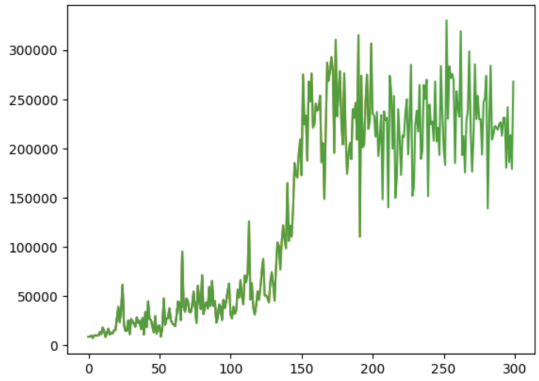

Figure 5. Performance with contact priority scoring function. The y-axis consists of the maximum score and the x-axis depicts the number of generations.

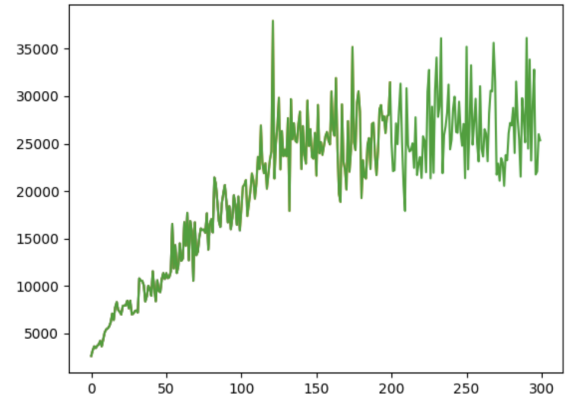

Figure 6. Performance with safe wall priority scoring function. The y-axis consists of the maximum score and the x-axis depicts the number of generations.

Figure 6 depicts the impact of the Safe Wall Priority scoring function across the next 300-generation window of training. As expected, there is a consistent accretion in the maximum achievable score by the agent. This window, sees a similarly improved score accumulation, leading to a substantial improvement before reaching a plateau. Ultimately, the agent's performance stabilizes, with the average score leveling off at around 35,000. At the end of this 300-generation window, the agent has learned to essentially avoid the polygon by swatting it by the arm along with resetting its trajectory to hit a safe wall. Finally, we subject it to a different set of rewards such that it does

not just swat it toward safe walls but is also able to ground it in the goal box.

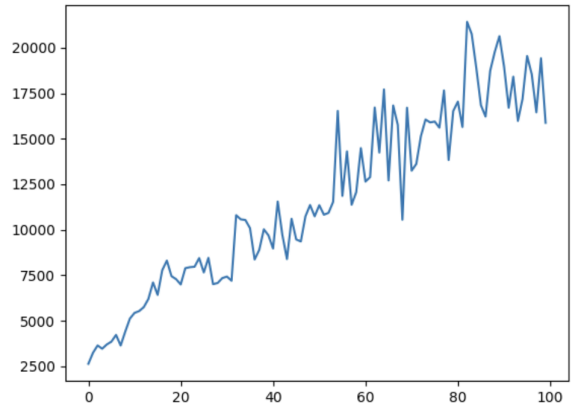

Figure 7. Performance with goal-box priority scoring function. The y-axis consists of the maximum score and the x-axis depicts the number of generations.

Figure 7 depicts the impact of the Goal-Box Priority scoring function across the final 100-generation window of training. At the end of this window, the agent has learned to essentially avoid the polygon by swatting it inside the goal box. The agent is thus optimized on various scoring functions over these generations and has learned to not only deflect the polygon but to do so with three different mechanisms.

To put into perspective the pros of using a variety of scoring functions, we depict the performance of the agent over 300 generations using a singular scoring function. This is depicted in Figure 8

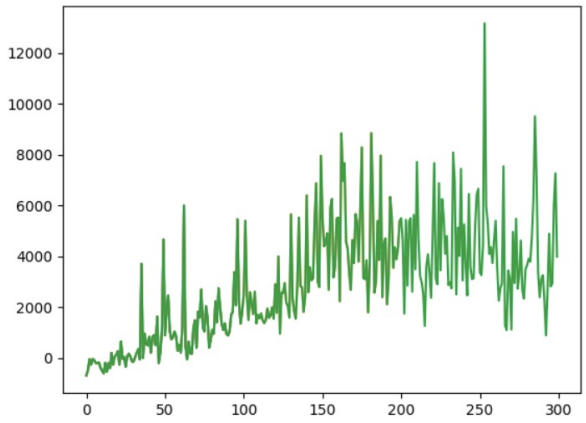

Figure 8. Performance with a single scoring function. The y-axis consists of the maximum score and the x-axis depicts the number of generations.

As noticed, the agent does not learn particularly well when compared to the average accumulation of maximum score that the agent receives. The performance levels out at about 7000 which is significantly lower compared to the performance metrics achieved after being optimized on the aforementioned scoring functions.

5. Conclusion

In this work, we have presented a model that has an agent that is trained gradually to deflect a polygon shot by a cannon from any angle to its goal state. We do this by training the agent in each generation for a given number of iterations, rewarding it for various factors, like making contact with the polygon, preventing it from hitting the target wall, in addition to deflecting it to the target state. It generates the score for these 'parents' in each iteration, selecting the top 10 parents with the highest score, combining them randomly, and mutating the next generation to produce children repeating the same process all over again. We train the model for at least 50 generations. We have used a genetic Algorithm for this.
In the above task, our model with the results displayed outperforms all previously reported ensembles.
We are excited about the future of this project as we place our agents in new environments (eg. no-gravity zones) and introduce more features like jitter, friction, lower rewards, etc. We plan to extend SHAPER in a 3-D space, re-orienting any shape and possibly with the ability to dodge obstacles. Making each generation less sequential is another research goal of ours.
Also, we wish to continue to explore the concept of slowly changing the score functions to reach specific milestones as this seems to produce good results in our specific use case. We also wish to explore if this idea can be incorporated into backpropagation. For example, changing the loss function during the training to explore how the training proceeds.

6. Team Contributions

The contributions to the code and the written submission is divided as follows -

1) **Ashwin Bharadwaj :**
   - Constructed the neural network and narrowed down the activation function. Along with deciding which spatial factors are necessary for the network to take into consideration.
   - Introducing Genetic Algorithm and genome mutation to pool in the best-performing agents with the maximum score to measure performance.
   - Introduction of process-based parallelism to allow training multiple agents in parallel to make the training times slightly more efficient.
2) **Aditya Ratan and Hasnain Sikora :**
   - Environment set-up and the creation of physics-based simulation using Pymunk and Pygame.

- Creation of the arm and actuating the collision handlers to manage polygon hits.
- Constraining the initialization of the cannon's angle and the polygon to ensure it angles itself toward hitting its target wall.

3) **Andrea Pinto :**

- Constructed scoring functions that allowed for an improved performance over subsequent generation windows.
- Introducing varied scoring functions with variable rewards.
- Ensuring that the results were properly delineated and that the agent reacted to the updated rewards as expected.
- Organized the appropriate graphs as well as edited sequences of the agent in action for written submissions.

7. Link to Code Repo

Our code written to train and evaluate the model is available at https://github.com/Its-a-me-Ashwin/SHAPER.git

References

[1] H. Chiroma, S. Abdulkareem, A. Abubakar, and T. Herawan, "Neural networks optimization through genetic algorithm searches: a review," *Appl. Math. Inf. Sci*, vol. 11, no. 6, pp. 1543–1564, 2017.

[2] T. Hermans, F. Li, J. M. Rehg, and A. F. Bobick, "Learning contact locations for pushing and orienting unknown objects," in *2013 13th IEEE-RAS international conference on humanoid robots (humanoids)*. IEEE, 2013, pp. 435–442.

[3] M. Islam, G. Chen, and S. Jin, "An overview of neural network," *American Journal of Neural Networks and Applications*, vol. 5, no. 1, pp. 7–11, 2019.

[4] A. Lambora, K. Gupta, and K. Chopra, "Genetic algorithm-a literature review," in *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 2019, pp. 380–384.

[5] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the genetic and evolutionary computation conference*, 2019, pp. 419–427.

[6] J. Shen, E. Xiao, Y. Liu, and C. Feng, "A deep reinforcement learning environment for particle robot navigation and object manipulation," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 6232–6239.

[7] J. Stastny and V. Skorpil, "Genetic algorithm and neural network," in *Proceedings of the 7th WSEAS International Conference on Applied Informatics and Communications*, vol. 7. Citeseer, 2007, pp. 345–349.

[8] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[9] S. Števo, I. Sekaj, and M. Dekan, "Optimization of robotic arm trajectory using genetic algorithm," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 1748–1753, 2014, 19th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016418653