Genetic Algorithm-Tuned Multilayer Perceptron for Self-Driving: EvoDrive Visualizer

Yong Ji Xiao
National Yang Ming Chiao Tung University
HsinChu City 300, Taiwan
caramelpudding0803@gmail.com

Abstract

This study introduces "EvoDrive Visualizer," a simulation framework driven by evolutionary computation to simulate and optimize autonomous car driving. The system utilizes a Genetic Algorithm (GA)[9, 11] to fine-tune a Multilayer Perceptron (MLP)[13], enabling cars to navigate tracks independently and adapt to diverse configurations.

The framework supports custom track environments and offers tools to automate track generation and promote generalization during training. Through real-time visualization and parameter adjustability, EvoDrive Visualizer demonstrates the effectiveness of evolutionary computation in flexible autonomous systems.

CCS Concepts: • **Computing methodologies** → **Genetic algorithms**; **Real-time simulation**; **Motion path planning**; *Neural networks*; • **Applied computing** → Transportation.

Keywords: Genetic Algorithm, Multilayer Perceptron, Evolutionary Algorithm, Autonomous Driving, Simulation Framework

ACM Reference Format:
Yong Ji Xiao. 2024. Genetic Algorithm-Tuned Multilayer Perceptron for Self-Driving: EvoDrive Visualizer. In *Proceedings of Evotionary Computation (EC).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Autonomous driving has emerged as a transformative technology, leveraging computational approaches to navigate vehicles in dynamic and complex environments. Traditional methods, such as supervised learning, often require labeled data and are constrained by predefined scenarios, limiting their scalability and adaptability.

This study builds upon prior experiments using Radial Basis Function Networks (RBFNs)[3] for self-driving applications. While effective in static environments, RBFNs rely heavily on labeled data for initializing center parameters, making them less versatile for broader use cases. To address this limitation, this project replaces RBFNs with Multilayer Perceptrons (MLPs)[13], which eliminate the dependency on labeled data and significantly enhance computational efficiency.

EvoDrive Visualizer was developed to further tackle the limitations of existing methods. The framework employs a Genetic Algorithm (GA)[9, 11] to optimize the parameters of an MLP, enabling cars to learn driving strategies independently. Unlike traditional optimization methods, which may become trapped in local optima, evolutionary computation explores diverse solutions through stochastic variations, balancing computational cost and adaptability.

The framework also introduces a simple track generator to reduce the complexity of environment design. This generator not only facilitates the creation of customizable tracks but also supports dynamic adaptation, where tracks are regenerated during training to evaluate the cars' generalization abilities. Through real-time visualization, users can monitor the training process in real time, providing insights into the effectiveness of evolutionary computation in autonomous systems.

2 Results

2.1 Training Process Visualization

EvoDrive Visualizer offers real-time visualization of the cars' evolution across generations. The interface is divided into sections that display the training process, fitness values over generations, the distribution of completion rates, and additional statistics such as the current generation and the highest fitness score.

For example, as shown in Figure 1, experiments conducted with different selection strategies, including Tournament Selection and Linear Ranking Selection, showed consistent improvements in overall fitness across generations. These results confirm the effectiveness of the GA in optimizing the performance of autonomous cars on varying track configurations.

EC, December 13, 2024, HsinChu City 300, Taiwan

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

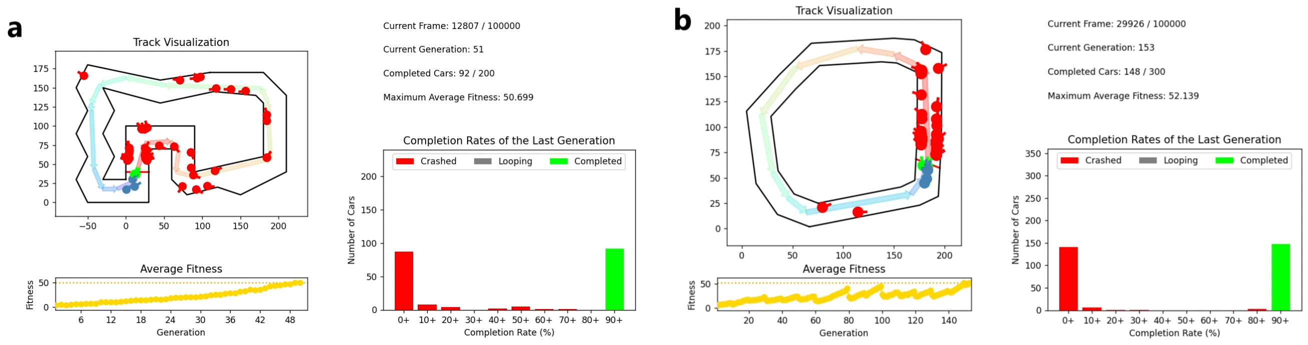

Figure 1. (a) Training process with Tournament Selection (tournament size = 2) on a fixed track. Average fitness rises rapidly in early stages and stabilizes later, showing a consistent upward trend. A 50% completion rate was achieved in approximately 25 minutes. **(b)** Training process with Linear Ranking Selection (selection pressure = 1.5) on tracks generated by the Simple Track Generator under Dynamic Adaptation Mode. Tracks changed every 10 generations, causing average fitness to drop temporarily but recover consistently. A 50% completion rate was achieved in approximately 1 hour.

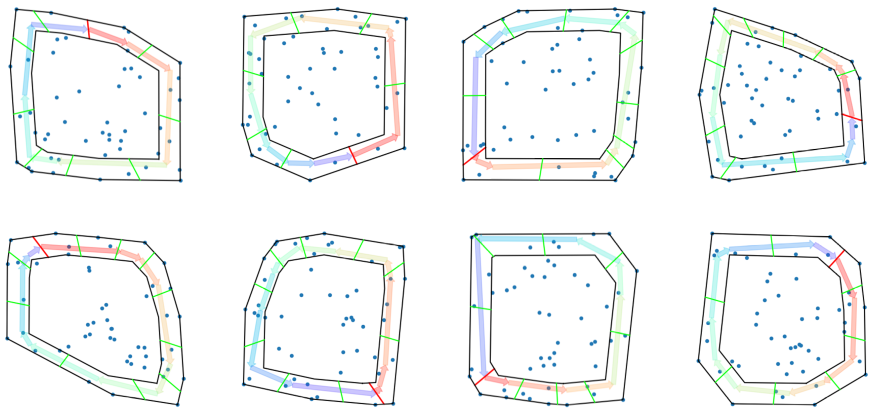

Figure 2. Tracks generated from eight runs of the Simple Track Generator, showing consistent structural patterns due to the use of convex hulls for both boundaries.

2.2 Simple Track Generator

Due to the difficulty for users to design a track manually, a feature was developed to automatically generate simple tracks. The algorithm generates random points within a constrained area and constructing a convex hull from these points to form the outer boundary of the track. The inner boundary is created by scaling the convex hull inward, followed by a validation process to ensure the paths are wide enough to ensure safe navigation for the car.

Although this feature significantly reduces the complexity of track design, the generated tracks tend to be simple in shape and lack diversity. This limitation arises because both the outer and inner boundaries are derived from identical convex hulls, resulting in configurations that are structurally similar (Figure 2).

2.3 Enhanced Usability Features

EvoDrive Visualizer includes several features to enhance its flexibility and usability.

2.3.1 Weights Preservation and Testing. Trained weights can be saved and loaded, allowing cars to be tested on new, unseen environments to evaluate their adaptability. For instance, weights trained on dynamic tracks can be applied to other track configurations, demonstrating the cars' ability to generalize and perform in diverse scenarios.

2.3.2 Dynamic Adaptation Mode. This optional mode is designed to ensure cars do not become overly specialized to a single track. During training, the track generator is periodically used to create new track configurations at regular intervals. By introducing variability in the training environment, this mode encourages cars to develop versatile driving strategies. When enabled, it enhances the generalization capability of cars, allowing them to adapt to a wide range of track layouts.

2.3.3 User-Adjustable Parameters. The framework allows users to customize a variety of parameters to suit different experimental setups. These include the size of the population, the architecture of the neural network's hidden layers, mutation rates, crossover probabilities, and the frequency at which new tracks are generated, among others. This flexibility enables researchers to tailor the training environment and algorithmic behavior, making the framework versatile for exploring different scenarios and objectives.

3 Discussion

EvoDrive Visualizer is a framework that leverages a GA to fine-tune an MLP, enabling autonomous cars to navigate circular tracks. Traditional optimization methods often suffer from significant limitations, such as getting trapped in local optima, which restrict their effectiveness in complex scenarios. EvoDrive Visualizer addresses these issues by utilizing evolutionary computation, demonstrating its robustness and adaptability in diverse environments. Additionally, the framework provides real-time visualization of the training process and allows users to customize training environments, including track configurations and neural network parameters. These features underscore its utility as a tool for exploring machine learning in autonomous driving.

The decision to transition from RBFNs to MLPs in this project was not solely due to performance limitations but also driven by the challenges associated with parameter tuning. RBFNs, while promising in specific applications, rely heavily on precise initialization of centers and associated parameters such as spread values. This dependency often demands significant effort in finding optimal configurations, which can hinder efficiency in broader scenarios. By contrast, MLPs reduce such dependencies, providing more adaptable network structures. However, the effectiveness of any neural network, including MLPs, still hinges on appropriate architecture and parameter choices. To address this, the framework allows users to customize hidden layer structures, enabling further exploration of network configurations and potentially accommodating other architectures with similar adjustments.

Another critical challenge lies in the design of track environments. Although the system supports obstacles and branching paths without dead ends, these features can affect the fairness and precision of fitness evaluation. Dead-end branches are particularly problematic due to the cars' inability to memorize their movement history, leading to situations where they become stuck. Furthermore, relying on a single approximate shortest path reduces evaluation accuracy in regions distant from this path, limiting the precision of fitness scores in more complex environments. Addressing these issues could further improve the system's adaptability and robustness.

4 Future Work

Although EvoDrive Visualizer demonstrates significant progress in applying evolutionary computation to autonomous driving, certain limitations highlight opportunities for future enhancements.

One key limitation is the fixed-speed assumption for cars during navigation. Currently, cars lack the ability to adjust their speed dynamically, which restricts their capacity to optimize both steering and traversal time. Expanding the MLP architecture to support speed variation could address this issue. By considering arrival time as part of the fitness evaluation would incentivize faster track completion and add new complexity to optimization.

Another challenge lies in the flexibility of genetic operators. While the current implementation supports basic operators such as selection, crossover, and mutation, it lacks the granularity needed for more specialized adjustments. Refining existing operators or introducing new ones could allow for more tailored experimentation, enabling researchers to investigate a wider range of evolutionary strategies and their impact on system performance.

Improving the realism of the simulation is another critical focus for future work. The system currently relies on simplified environmental conditions, which reduces its relevance to real-world applications. To address this, incorporating more complex elements like varying terrains, dynamic obstacles, and a physics engine could make the simulation more accurate. Adding sensors capable of detecting velocity, acceleration, or terrain properties would further enhance the system's ability to replicate real-world driving scenarios, ultimately narrowing the gap between simulation and practical deployment.

Enhancing computational efficiency is a key consideration for future improvements. As the population size grows, the training process becomes increasingly resource-intensive, causing slower visualizations and extended computation times. Similarly, larger and more complex tracks demand greater processing power, further straining the system's efficiency. Implementing parallelization techniques, such as distributed processing or multi-threading, could address these challenges by accelerating calculations and maintaining efficient performance under high computational demands.

Lastly, while the current track generator simplifies environment creation, its output remains relatively basic. Generating tracks with varying widths, intersecting paths, or dynamic obstacles could provide a more comprehensive evaluation of the cars' adaptability. These enhancements would further demonstrate the potential of evolutionary computation to handle diverse and challenging scenarios.

EvoDrive Visualizer has successfully demonstrated the potential of evolutionary computation in autonomous driving, providing a flexible and customizable framework for training and evaluating adaptive strategies. The system's emphasis on real-time visualization and user-configurable features establishes a strong foundation for both practical applications and academic exploration. Looking ahead, addressing current limitations and incorporating proposed enhancements will not only improve usability but also expand the system's flexibility and scalability, enabling it to adapt to a wider range of use cases and evolving research needs.

References

[1] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517. https://doi.org/10.1145/361002.361007

[2] Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches* (San Diego, California) *(SIGGRAPH '07)*. Association for Computing Machinery, New York, NY, USA, 22–es. https://doi.org/10.1145/1278780.1278807

[3] David S. Broomhead and David Lowe. 1988. Multivariable Functional Interpolation and Adaptive Networks. *Complex Syst.* 2 (1988). https://api.semanticscholar.org/CorpusID:3686496

[4] E.W. DIJKSTRA. 1959. A Note on Two Problems in Connexion with Graphs. *Numer. Math.* 1 (1959), 269–271. http://eudml.org/doc/131436

[5] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*. https://api.semanticscholar.org/CorpusID:5575601

[6] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[7] David E. Goldberg and Kalyanmoy Deb. 1991. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundations of Genetic Algorithms*, GREGORY J.E. RAWLINS (Ed.). Foundations of Genetic Algorithms, Vol. 1. Elsevier, 69–93. https://doi.org/10.1016/B978-0-08-050684-5.50008-2

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 1026–1034. https://doi.org/10.1109/ICCV.2015.123

[9] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press. https://doi.org/10.7551/mitpress/1090.001.0001

[10] Tomás Lozano-Pérez and Michael A. Wesley. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM* 22, 10 (Oct. 1979), 560–570. https://doi.org/10.1145/359156.359164

[11] Zbigniew Michalewicz. 1994. *Genetic algorithms + data structures = evolution programs (2nd, extended ed.)*. Springer-Verlag, Berlin, Heidelberg.

[12] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Haifa, Israel) *(ICML'10)*. Omnipress, Madison, WI, USA, 807–814.

[13] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323 (1986), 533–536. https://api.semanticscholar.org/CorpusID:205001834

A Research Methods

A.1 Defining the Genetic Algorithm Framework

To apply the Genetic Algorithm (GA)[9, 11], several essential components must first be defined to guide the optimization process.

Individual. Each individual in the population represents a car, independently controlled by its own Multilayer Perceptron (MLP)[13]. The car's five sensors measure distances to walls in different directions, providing input to the MLP for navigation decisions. The MLP consists of an input layer (5 neurons), customizable hidden layers, and an output layer (1 neuron) that determines the car's steering angle (Figure 3). The MLP's weights and biases form the genotype, while the car's navigation behavior represents its phenotype.

Population. The population is composed of a user-defined number of individuals. Its size is determined at the start of the training process and remains fixed throughout all generations. This design choice prevents computational overhead while ensuring the algorithm maintains sufficient genetic diversity for effective optimization.

Weights Initialization. The weights of the neural network are initialized differently depending on the layer type to ensure stable gradient flow and effective training for each layer type (Figure 3). All biases are initialized to 0 to ensure consistent network behavior. For the hidden layers, *He Initialization*[8] is applied to complement the *ReLU*[12] activation function, optimizing variance flow during training. The output layer uses *Xavier Initialization*[5], which is well-suited for *Sigmoid*[13] activation to maintain stable gradients. These initialization methods are tailored to the activation functions used in each layer and are mathematically defined as follows:

- $W_{\text{hidden}} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}}\right)$
- $W_{\text{output}} \sim \mathcal{N}\left(0, \frac{2}{n_{\text{in}}+n_{\text{out}}}\right)$

Here, n_{in} represents the number of neurons feeding into the layer, and n_{out} represents the number of neurons in the subsequent layer.

A.2 Genetic Algorithm Operations

GA iteratively optimizes the weights of the neural networks through a series of evolutionary operations. These operations ensure a balance between exploration of the solution space and exploitation of high-performing solutions. The key operations are described below:

Selection. This process determines which individuals are chosen as parents for the next generation, ensuring that higher-performing individuals have a greater chance of contributing to the population's evolution[6, 7]. The framework implements three strategies:

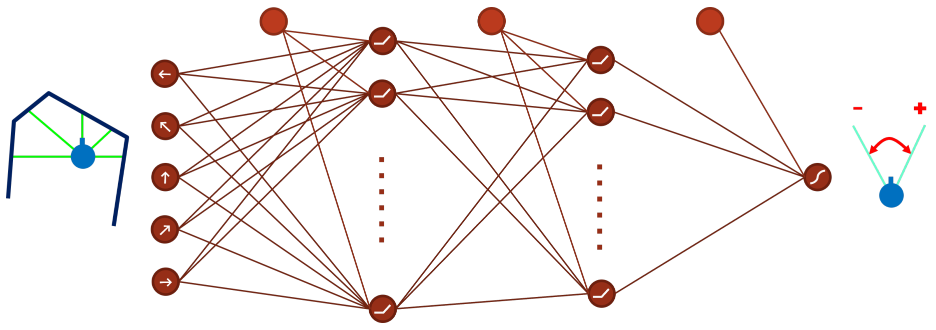

Figure 3. Illustration of the MLP architecture used to control the car. The input layer processes sensory data representing distances to surrounding walls, and the output layer determines the car's steering direction. The structure includes input, hidden, and output layers, with trainable weights and biases connecting all layers.

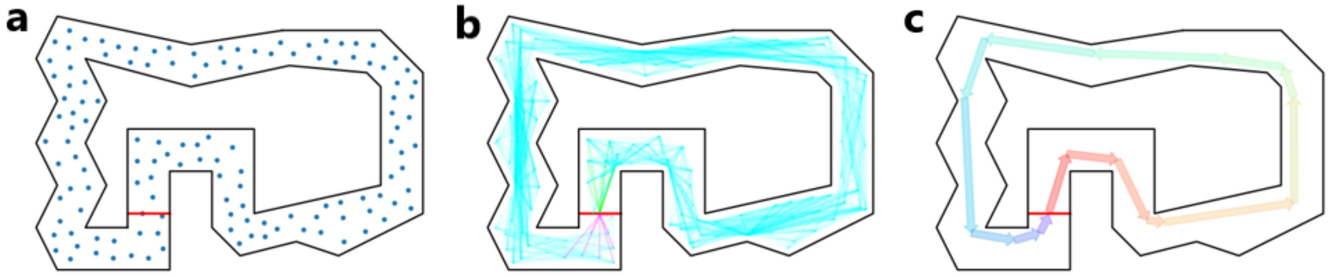

Figure 4. Pathfinding process for establishing the ASP: **(a)** Points generated within the track area using Poisson Disk Sampling, ensuring even distribution and sufficient spacing to cover the track efficiently. **(b)** Valid connections between sampled points, forming the edges of the Visibility Graph. Each edge represents a direct line of sight that does not intersect track boundaries. **(c)** The ASP computed from the Visibility Graph using Dijkstra's Algorithm, representing the optimal route from the start line to the finish line.

- **Roulette Wheel Selection**: Individuals are selected with probabilities proportional to their fitness scores. For individual i, the probability $P(i)$ is defined as:

$$P(i) = \frac{f_i}{\sum_{j=1}^{N} f_i}$$

  where f_i is the fitness of individual i, and N is the total number of individuals.
- **Linear Ranking Selection**: Individuals are ranked by fitness, and selection probabilities are assigned based on ranks. For individual i, ranked r_i, the probability $P(i)$ is computed as:

$$P(i) = \frac{2-s}{N} + \frac{2r_i(s-1)}{N(N-1)}$$

  where s is the selection pressure, which is set to a default value of 1.5, and N is the population size.
- **Tournament Selection**: A fixed number of individuals are randomly chosen to compete, and the one with the highest fitness is selected. The default tournament size is 2.

Two individuals are selected in each iteration to serve as parents for subsequent crossover and mutation operations.

Crossover. This operation facilitates the exchange of genetic information between two parents to produce offspring. Each weight in the neural network has a 50% probability of being exchanged with the corresponding weight of the other parent.

Mutation. Each offspring's weights are subject to mutation, introducing random changes to mutation prevents stagnation by introducing random variations into the solution space. This process involves a 20% probability of reinitializing each weight in the neural network, based on the following schemes:

- Hidden layers: **He Initialization**[8].
- Output layer: **Xavier Initialization**[5].
- Biases: Always reinitialized to 0.

Elitism. This strategy preserves only the cars that successfully reach the finish line. These elite individuals remain unchanged until the track configuration changes, ensuring that high-performing solutions are not overwritten by random genetic operations.

Elimination. An optional elimination mechanism can be applied to remove a specified proportion of the individuals with the lowest fitness scores. This step accelerates convergence by reducing the impact of poorly performing individuals but is not required in every experiment. Notably, applying elimination must balance the need for faster convergence with maintaining sufficient diversity to ensure effective optimization.

A.3 Fitness Evaluation

Fitness evaluation is a critical component of the GA, as it determines the performance of individuals and drives the evolutionary process. In this framework, fitness is calculated based on each car's progress along a predefined approximate shortest path (ASP). To achieve this, the first step involves generating an ASP for the given track, which serves as a baseline for evaluation. The process involves pathfinding and fitness scoring.

A.3.1 Pathfinding. To establish a reference for evaluation, the algorithm computes the ASP for the given track. The process consists of three main steps (Figure 4):

1. **Point Sampling**: *Poisson Disk Sampling*[2] is used to generate evenly distributed points across the track area. This method ensures uniform coverage while avoiding overly close points, with a KDTree[1] employed to efficiently check point distances, improving sampling efficiency for irregularly shaped tracks.
2. **Graph Construction**: A *Visibility Graph*[10] is constructed from the sampled points. This undirected weighted graph connects nodes with edges that represent direct lines of sight, ensuring they do not intersect track boundaries. Additionally, in the circular track, the start and finish lines overlap; these points are slightly offset based on the initial direction to prevent ambiguity.
3. **Path Calculation**: *Dijkstra's Algorithm*[4] is applied to the Visibility Graph to compute the shortest path. The graph's edge weights represent the Euclidean distances between connected points. As the weights are positive, Dijkstra's Algorithm is an appropriate choice.

The resulting ASP establishes a reliable reference for evaluating each car's progress along the track.

A.3.2 Fitness Scoring. The fitness of each car is evaluated based on its performance relative to the computed ASP. Scores are normalized within a range of 0 to 100, using the total length of the ASP as the reference for scaling. The evaluation process considers three primary scenarios:

1. **Completion**: Successful completion of the track results in a maximum fitness score of 100. To further distinguish between completed and incomplete attempts, a multiplier bonus is applied, emphasizing the advantage of successful completion.
2. **Crash Handling**: For cars that crash into walls or obstacles, their final position is projected onto the ASP. The fitness score is calculated as the percentage of the ASP completed before the crash, providing a fair evaluation for incomplete attempts.
3. **Looping**: Cars that fail to make meaningful progress and continue looping indefinitely receive a fitness score of 0. This ensures that cars exhibiting unproductive behaviors, such as looping indefinitely, are penalized.

These fitness scores guide the selection process in subsequent generations, ensuring that high-performing cars contribute more significantly to the population's evolution.

B Online Resources

The source code for EvoDrive Visualizer, including the implementation of the described algorithms, is available at: https://github.com/pudding0803/EvoDrive-Visualizer.