Nature Inspired Computing for Gaming: Evolutionary Neural Network for game AI Algorithm

Grigorii Fil
Innopolis University
Email: g.fil@innopolis.university

Konstantin Fedorov
Innopolis University
Email: k.fedorov@innopolis.university

Mostafa Khaled Mostafa
Mohamed Aref Kira
Innopolis University
Email: m.kira@innopolis.university

1. Introduction

Artificial intelligence (AI) is vital part of games because smart AI can improve game experience, while unintelligent AI leads to frustration of players. Classical AI, which is rule based, is common approach for game developers. However, for creating a smart AI, complicated nested conditions and deep understanding mechanics are required. To simplify this process we can use Neural Networks which showed better performance in most cases. Furthermore, in some games they are comparable to humans or even better. For example, AlphaStar bot for StarCraft 2 was above 99% of ranked humans players (Vinyals et al. [3]). Overall, our goal is to find efficient method of training Neural Network based AI for games.

For training Neural Networks in games popular methods are reinforcement learning and evolutionary algorithms. For reinforcement learning for every action immediate reward should be calculated. However, in some games only final reward can be computed. On the other hand, evolutionary algorithms can work with this type of reward. Thus, we will use Evolutionary algorithms for training Neural Networks.

To test our methods we will use Gymnasium environment 'Car Racing' [2]. Gymnasium was chosen because of simple interface and plenty available games. The 'Car Racing' environment is a top down racing game. Its observation space is 96 by 96 RGB image, which includes top down view of track and user interface with information about speed, ABS, steering angle and angular acceleration of a car. The environment's action space is 3 numbers: steering (number from -1, full turn to the left, to 1, full turn to the right), gas (from 0, not accelerate, to 1, full acceleration), and breaking (from 0, no breaking, to 1, full breaking).

2. Related Work

In the paper [4] same environment was explored using Genetic Algorithm with Neural Networks. Weights and biases of neural network are encoded as genes in a chromosome. Therefore, structure of the network remains the same. Mutation is done in the following manner: random genes are taken and new normal distributed random values are assigned. Crossover exchanges random sequence of genes between chromosomes. Overall, this approach gave reward between 856 and 872. Thus, Neural Network with fixed layer architecture seems as promising method.

Another source of our inspiration is paper [6]. Where different neuroevolution approaches are tested on a 61 Atari games. One of the models is NeuroEvolution of Augmented Topologies (NEAT). One of the inputs types is downscaled game frame, which showed acceptable performance. Therefore, we can also use this technique.

One of the problem with approach [4] is fine tuning of Neural Network architecture. In the paper [5] NeuroEvolution of Augmented Topologies (NEAT) method described. This approach has a lot of advantages. Firstly, structure of neural network changes during the process of evolution. Secondly, structure of node is similar to directed graph and not limited by layers. Finally, all networks begin with small amount of nodes, then gradually increase their complexity. That what makes NEAT produce concise networks. That is why we decided to use NEAT.

3. Methodology

The model for the game will take the image as an input (Figure 1, top left image). As the picture is big (96x96 pixels in RGB format, 27648 values total), we decided to make preprocessing for feature extraction using openCV library. We used two approaches: Raycasting and Binary Image.

For the model we will use Fully Connected ANN and NeuroEvolution of Augmenting Topologies (NEAT).

3.1. Preprocessing: Binary Image

To begin with, position of car is static for every game frame. Also, road, car and grass has constant RGB colors. All of preprocessing approaches starts with removing bottom part of frame where UI is. After that remaining image is binarized using prior knowledge of road's pixel rgb (103, 103, 103). Every pixel with row index i and column index j has its $r_{i,j}$, $g_{i,j}$, $b_{i,j}$ representing red, green and

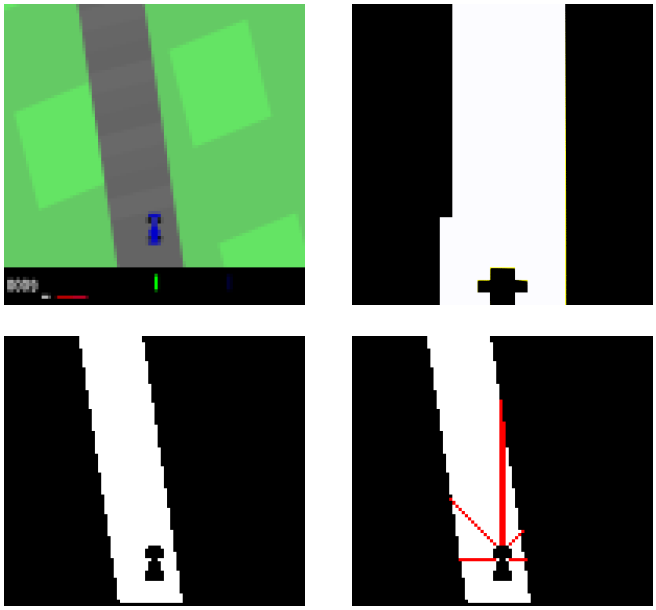

Figure 1. Top left is original frame, Bottom left is binarized without UI, Top right is Binary preprocessing method, Bottom Right is demonstration of rays

blue channel respectively. Pixel $p_{i,j}$ of binary image can be calculated based on Manhattan distance to road pixel as:

$$\begin{cases} b_{i,j} = 255, \ |r_{i,j} + g_{i,j} + b_{i,j} - 309| < 10 \\ b_{i,j} = 0, \text{ otherwise} \end{cases}$$

Here 309 is a sum of road's pixel rgb and 10 is our threshold. Result of binarization can be seen on Figure 1 in bottom left corner.

Binary image approach takes inspiration from [6] where downscaled frame of actual game gave acceptable result. In our method we will make 48 by 48 center crop of initial RGB frame, then rescale it to 24 by 24 and binarize it. This gives us binary 24 by 24 frame (Figure 1, top right image). Then we normalize it from 0 to 1.

3.2. Preprocessing: Raycasting

Raycasting in this context is distance to closest given pixel from some point in some direction. Raycasting applied to already binarized frame. There are 2 types of rays: until grass pixel and until road pixel. First type is needed for viewing on the road while second needed for cases when car gets off track. We calculate rays in 5 direction: forward, left side, right side, left $45°$, right $45°$. Starting positions rays are given it table below.

Ray direction	x	y
forward1	47	66
forward2	48	66
left side	46	70
right side	49	70
left 45	46	66
right 45	49	66

There are 2 forward rays because number of car pixels is even. Final forward distance is calculated as minimal of this two. Overall 10 rays are computed: 5 until grass and 5 until road. Example of until grass rays are shown in Figure 1 bottom right.

Furthermore, speed, steering angle and angular acceleration are parsed from bottom panel of UI. For speed UI we know that speed pixel is white and its lowest pixel coordinates. Then, we can go 'up' in the frame until we met not white pixel and count how much pixels we passed. Next, we divide this value by maximum possible speed to normalize it. In the similar manner we can do with steering angle and angular acceleration but instead of goin 'up' it will be 'left' and 'right'.

To sum up, output vector will have size 13 and will include: speed, steering angle, angular acceleration, 5 rays until grass and 5 rays until road.

3.3. Model: Fully Connected ANN

For this model, we decided to use only Raycasting preprocessing technique, because Binary Image technique produces many features (576). Through the fine-tuning process we obtained such architecture:

Layer	Number of Neurons
Input	13
Hidden1	8
Hidden2	6
Hidden3	6
Output	2

For every layer except the last one ReLu was used as an activation function. For an output activation we used the sigmoid scaled by a factor of two and shifted down by one (Figure 2). Thus, our model will return two values from -1 to 1. The first value is forwarded to the steering action of the environment. If the second value is positive, then it is forwarded to the gas action and breaking action is set to 0. If it is negative - then its absolute value is forwarded to breaking action and gas is set to 0.

Initial population is generated with random weights drawn from normal distribution with mean 0 and standard deviation 2. The number of individuals is set to 16.

During each mutation, we select one random layer of a model for updating. Other layers can mutate during the same mutation with probability 0.3. Inside each layer, we choose 2 weights and update them with the values drawn

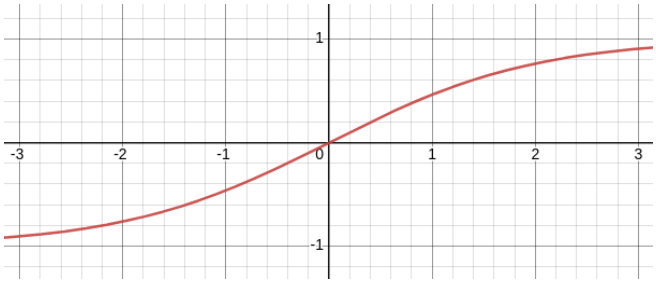

Figure 2. Output activation function

from normal distribution with mean 0 and standard deviation 2.

The crossover is simple. First model replaces weights of one of its layers by weights of the layer from second model, and vice versa.

At each evolution step, we divide our population in pairs (1st place with 2nd place, 3rd with 4th, and so on) to make crossover. Then, we perform mutation for each individual (for each parent and for each offspring), but retain the initial generation. And finally, we test each model on randomly generated environment (same environment for each individual) and choose best 16 (population size) for the next generation.

For fitness score we used reward provided by gymnasium. The reward is decreased by 0.1 every frame and increased by 1000/N for each tile visited by car. Maximum possible reward is 1000. If an agent does not score any points withing 50 frames, we terminate the race beforehand to speed up the learning process. Also, the model does not make decision every frame. To hasten the learning process more, model performs preprocessing and chooses an action once per three frames. When the model does not make decision, action is set in the following way: steering is halved from the previous action, gas and breaking is set to 0.

3.4. Model: NEAT

For the NEAT model, both preprocessing techniques were used and same fitness score was applied. Let briefly describe how NEAT works. NEAT is genetic algorithm that tune structure and weights of neural network in form of directed graph. Every node of graph is one perceptron. Gene consists of following data: source node, destination node, weight and innovation number. Innovation number is a global integer value that initially equal to 0.

Mutation consist of 2 types: weight and structural. While weight mutation is not different from others approaches, structural mutation is reason why this algorithm can tune its complexity. It is done in two ways: adding new connection and adding new node. In first way we add new connection between 2 existing nodes. In second way a new node is added instead of connection between 2 existing nodes. The new node will have connections to these 2 existing nodes and one connection will have weight 1 and other will have weight equal to weight of replaced connection. This is done to minimize loss of fitness for added new connection. Because it requires some iterations to mutate for obtaining proper weight. Also, for every added connection current global innovation number is assigned. Then, innovation number is increased by 1.

Population in NEAT split among species based on their structural similarity. That's similarity is calculated on how many genes have same innovation number. Crossover is performed only between similar species.

Main advantage of this method is gradual increase of complexity. Thanks to this property concise neural nets are usually produces.

As an implementation of neat we used open source library neat-python [7]. Our changed parameters were there were 0 initial connections and 0 zero hidden nodes to produce consise neural nets. Whole config you can see in our github [1].

4. Experiments and Evaluation

We evaluated each model using gymnasium environment and used reward as a metric. We also considered the fact of finishing the race. In the table below, we present the average fitness score, maximum achieved fitness, and percent of completed races. Note that unfinished races were not used to calculate the average fitness.

Model	Preprocessing	Avg Fitness	Max Fitness	Finished
FC ANN	Raycasting	850	873	75%
NEAT	Binary Image	830	866	85%
NEAT	Binary Image	870	889	95%

5. Analysis and Observations

From our experiments we concluded that NEAT model with Raycasting approach performs better than other models. It has the best fitness score and can complete most of the races. Fully Connected ANN has fitness score greater than the score of NEAT with Binary Image approach, because it can complete the race faster and scores less penalty. Although, Fully Connected ANN is more prone to error and gets off track more often than NEAT.

Another point to mention is that the structure of ANN model must be tuned by hand. This process can be slow and inefficient. But NEAT model can evolve the structure by itself. It can decide by itself which connections are need through the process of evolution. Final model with Raycasting approach is shown in the Figure 3. From this graph we can make following analyze. Firstly, most of the inputs are not used at all. Secondly, there is only one hidden node that affects output. Finally, we can make guess how NEAT model makes decision. Steering is mostly affected negatively by until grass left $45°$ ray and positively by until grass right $45°$ ray. In straight line sum of this rays will

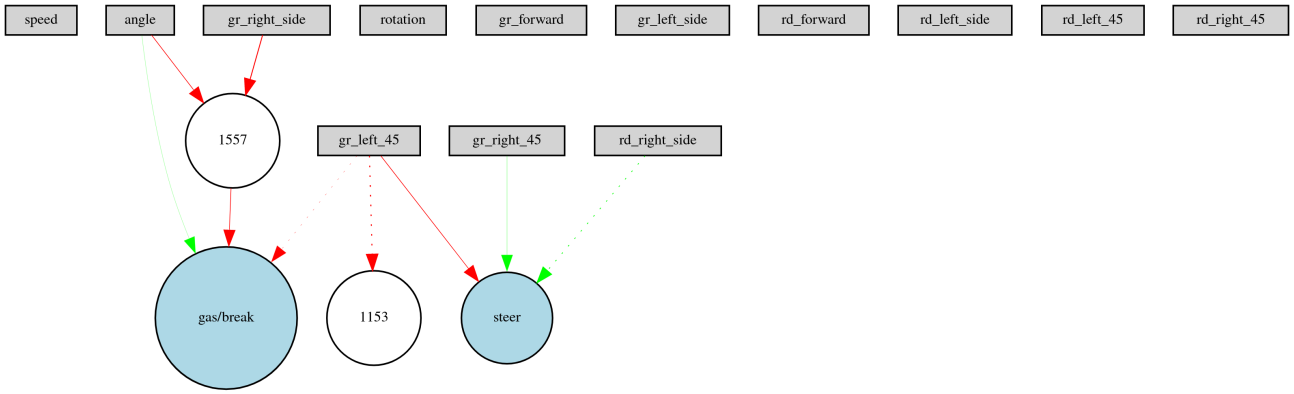

Figure 3. NEAT final structure

give 0, but in case of left or right corner corresponding ray will increase dramatically and it will steer in this direction. However, gas/break policy is not so easy to understand.

6. Conclusion

In conclusion, approach with passing downscaled image showed good performance considering little efforts on preprocessing. On the other hand, Raycasting method showed better results but it required sophisticated preprocessing.

Fully Connected ANN showed good performance. Also, the ANN has large structure that requires fine tuning. On the other hand, NEAT model outperformed the ANN. In addition, algorithm can handle excessive inputs. Furthermore, resulting model is concise and light weighted. To sum up, we can say that NEAT is a great method for creating AI in games.

The code for this project can be found on github page [1].

References

[1] https://github.com/KonstFed/CarRacing-Evolutionary-Algorithm

[2] Oleg Klimov. "Car Racing." https://gymnasium.farama.org/environments/box2d/car_racing/ (accessed April 22, 2023).

[3] Vinyals, O., Babuschkin, I., Czarnecki, W.M. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 575, 350–354 (2019). https://doi.org/10.1038/s41586-019-1724-z

[4] Li, Changmao. "Challenging on car racing problem from openai gym." arXiv preprint arXiv:1911.04868 (2019).

[5] Stanley, Kenneth O., and Risto Miikkulainen. "Evolving neural networks through augmenting topologies." Evolutionary computation 10.2 (2002): 99-127.

[6] Hausknecht, Matthew, et al. "A neuroevolution approach to general atari game playing." IEEE Transactions on Computational Intelligence and AI in Games 6.4 (2014): 355-366

[7] https://github.com/CodeReclaimers/neat-python (accessed May 1, 2023).