National Institute of Technology Calicut

EE3026D ANN and Fuzzy Logic Systems Course Project

Training an AI to play “flappy bird” using N.E.A.T

Anirudh V Nair

B200300EE

Abstract –

This project focuses on creating an AI that can play the game “Flappy bird” for an indefinite (or infinite) duration in python. For this, we use an evolutionary algorithm known as NeuroEvolution of Augmenting topologies (or N.E.A.T for short). We’ll also be using python “pygame” module for creating the base game.

We shall first understand the basics and the configuration file for N.E.A.T and then look into the chosen input parameters, fitness function and various changes we made while training the AI.

The N.E.A.T algorithm

“NeuroEvolution of Augmenting topologies” is a genetic algorithm used for the generation of artificial neural networks that are capable of evolving. It was developed by Kenneth Stanley and Risto Miikkulainen in 2002 at The University of Texas, Austin.

It resembles Charles Darwin’s principle of “natural selection” where the species evolves due to change in heritable traits in a population occurring because of differential survival and reproduction of individuals due to difference in phenotypes. The N.E.A.T algorithm favours individuals with more survival capability or fitness value, and allows their speciation to preserve their qualities and also improve survival chances in succeeding generations.

The N.E.A.T algorithm initially creates a population with random features and zero fitness value. When the individuals succeed in the required aspects, we reward them by increasing their fitness value. At the end of a generation, a certain percentile of individuals is chosen who have a high fitness value. The properties of these individuals are interbred to create the next generation. This process continues until an individual with highest survival chances is born.

The neural network initially begins with a simple perceptron like feed-forward network with only input and output neurons. But as

generations go by, the neural network becomes increasingly complex with creation, addition and deletion of hidden networks and nodes.

The N.E.A.T configuration file

We shall now look into the configuration file which is needed to implement the N.E.A.T algorithm and understand certain terminologies which are to be adjusted for this project.

[NEAT] Section –

1. *"fitness_threshold"* – When the fitness value being calculated meets or exceeds this value the evolution process is terminated and the program stops. We have set it to be 5000.
2. *"pop_size"* – The number of individuals (population) in a generation. We have set it to 100.

[DefaultGenome] section –

1. *"activation_default"* – The activation function to be used. Here we have used the tanh function.
2. *"activation_mutate_rate"* – Probability that the activation function will mutate/change. We have set it to be 0.
3. *"activation_options"* – List of activation functions which can be used for mutation. We have only specified tanh here.
4. *"bias_max_value"* – Maximum possible value for bias. We have set it to be +30.
5. *"bias_min_value"* – Minimum possible value for bias. We have set it to be -30.
6. *"feed_forward"* – True/False to enable or disable the feed forward network.
7. *"num_hidden"* – Number of hidden nodes initially present. We have set it to be zero initially.
8. *"num_inputs"* – Number of input nodes. We set it as 3 for initial testing and then later changed to 4.
9. *"num_outputs"* – Number of output nodes. We have set it 1 as we need only one output.

10. "weight_max_value" – Maximum allowed value for weights. We have set it to +30.

11. *"weight_min_value"* – Minimum allowed value for weights. We have set it to -30.

[DefaultStagnation] section –

1. *"max_stagnation"* – Species that show no improvement with these number generations will be considered stagnant and will be removed. We have set it to 20.

[DefaultReproduction] section –

1. *"survival_threshold"* – The fraction of species which will be allowed to reproduce each generation. We have set it as 0.2.

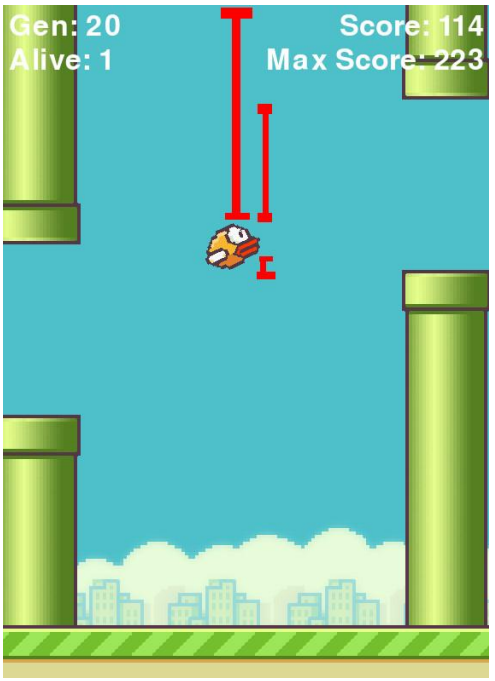
Other Libraries Used

1. **Pygame -** It is a module used to create video games in python. We used this module to create the base "flappy bird" game.
2. **os -** We used it to show the path to the various images required to create the base game.
3. **matplotlib.pyplot -** We used it to plot all the necessary graphs.
4. **visualize -** It is an external python code provided in the official docs page, used to create certain graphs involving fitness and other values. We have to save the file 'visualize.py' in the same folder as the main project file before importing it.

Initial Testing

A. Input Parameters -

We decided to use three distance parameters, namely - y position of bird, absolute value of difference between y coordinate of the bird and the bottom of the top pipe and difference between y coordinate of the bird and the top of bottom pipe, as the input parameters for the neural network.




```
#Providing I/P parameters to neural network
output = n_net[x].activate((bird.y, abs(bird.y - pipes[pipe_index].height), abs(bird.y - pipes[pipe_index].bottom)))
```

B. The Fitness Function -

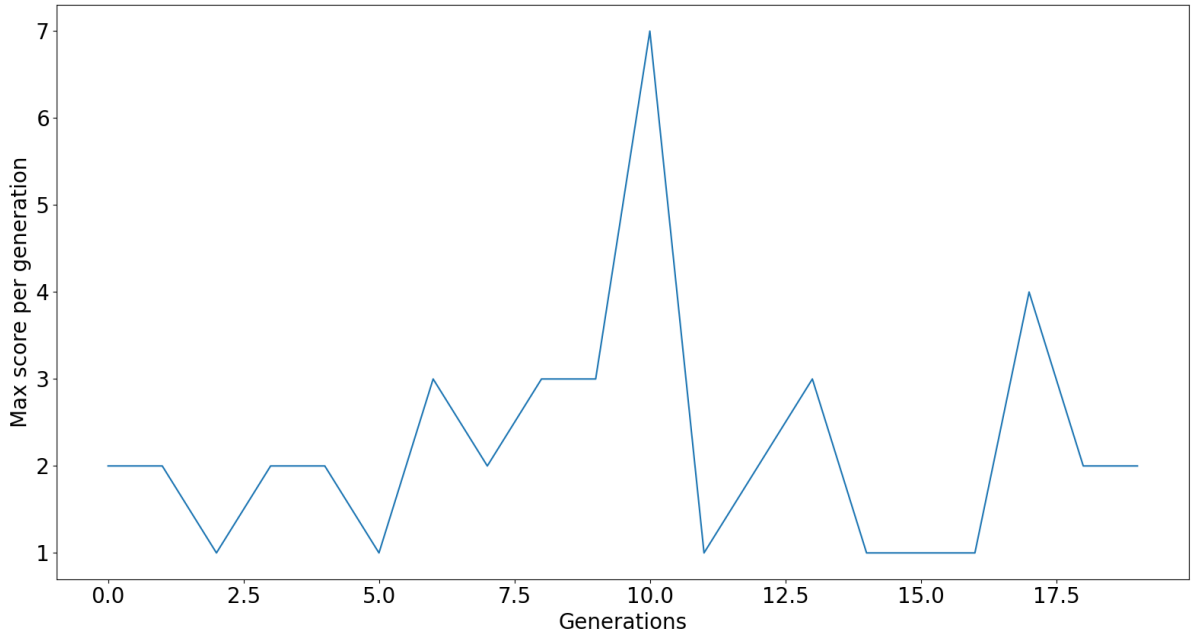
We will be rewarding the AI as and when it achieves some goals and also punish it accordingly.

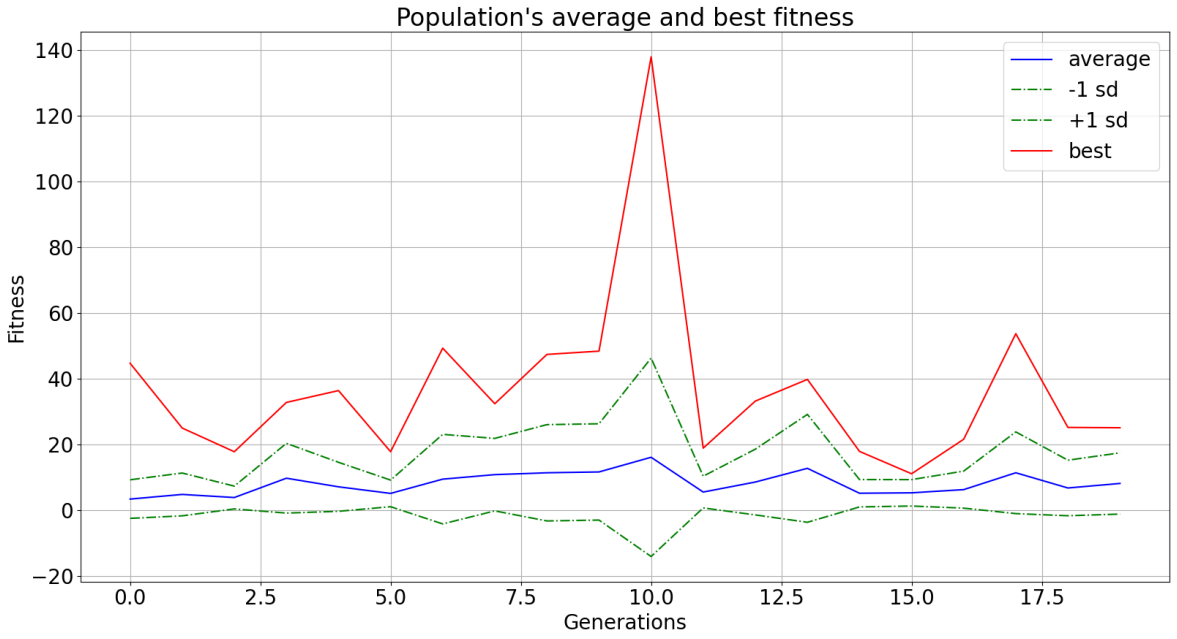
1. For every frame it's alive, we will award it +0.1 fitness.
2. Everytime it collides with a pipe, we will punish it with -5 fitness.
3. Everytime it passes through a pipe (i.e scores a point), we award it with +5 fitness.

4. Everytime it surpasses the previously achieved max score, we award it with +10 fitness.

C. Output -

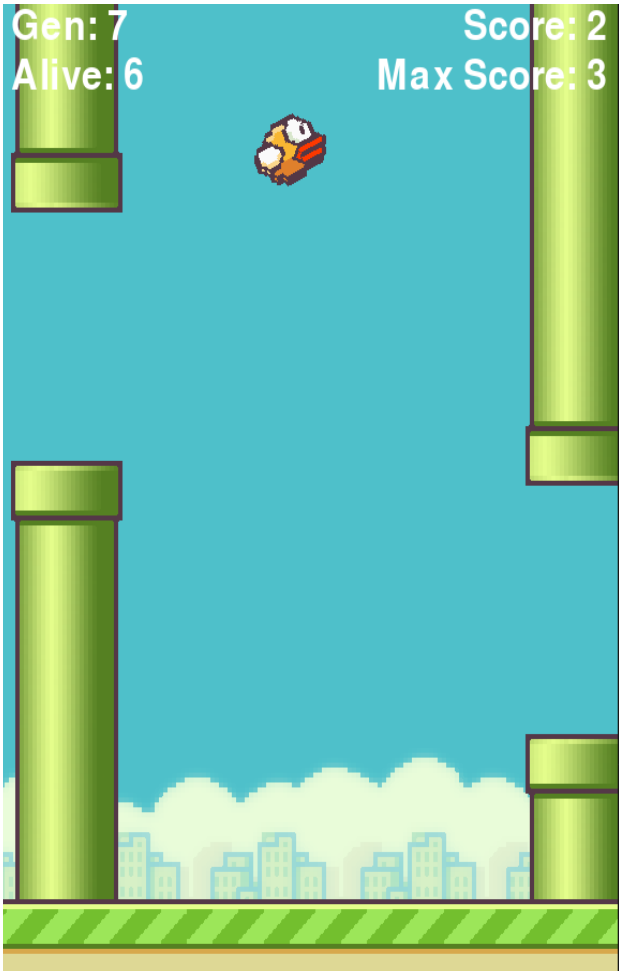
For the aforementioned inputs and fitness function, we achieved the following outputs after 20 generations with a population size of 100.





As we can observe, the performance of the AI is very poor (Max score being 7). This is mainly because the AI is unable to cover large

differences in height and at times keeps jumping up even though it should be coming down (as seen in the picture below).



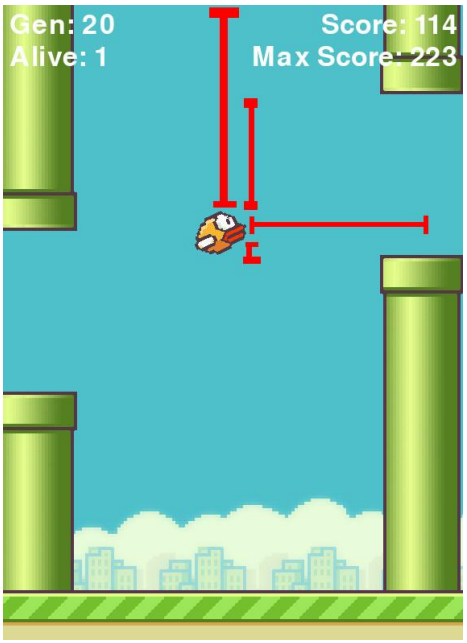
This defect is retained by each generation as a result of which none manage to survive for a longer time period.

The result doesn't change with increasing the number of generations or population size and persists even when we try a new simulation. Thus, some corrections were needed in our approach.

Corrections and Final Results

A. Input Parameters -

This time we took four input parameters instead of three, namely - y position of the bird, difference (not absolute value) between y coordinate of the bird and bottom of top pipe, difference (not absolute value) between y coordinate of the bird and top of bottom pipe, absolute value of horizontal distance between bird and pipes.



Now the bird can know if the space between pipes is above or below it. It also can adjust its vertical position depending on the horizontal distance between the pipe and itself.

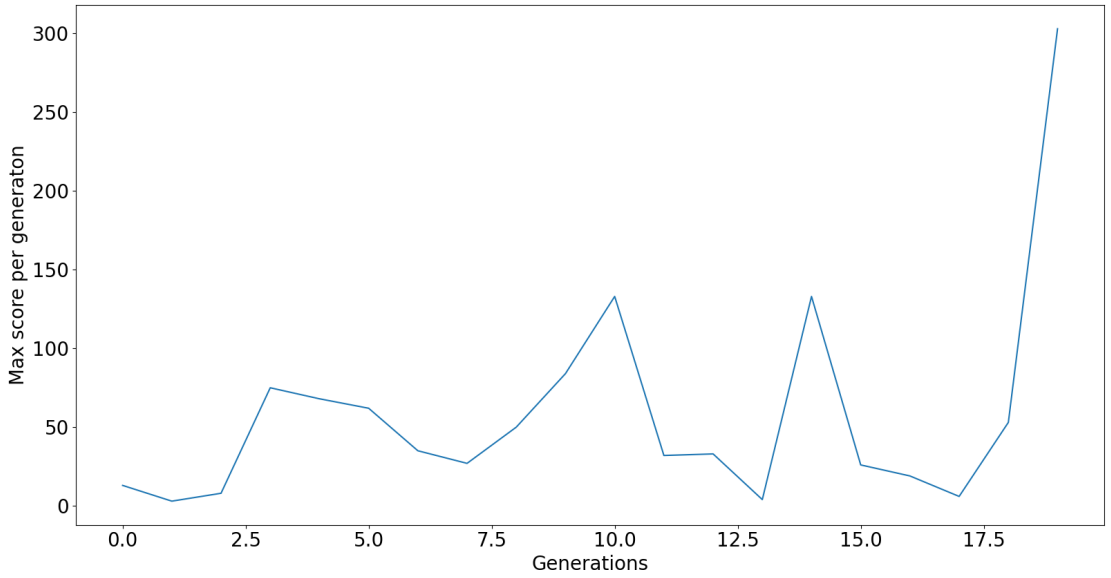
```
#Providing I/P parameters to neural network
output = n_net[x].activate((bird.y, (bird.y - pipes[pipe_index].height),
                            (bird.y - pipes[pipe_index].bottom), abs(bird.x - pipes[pipe_index].x)))
```

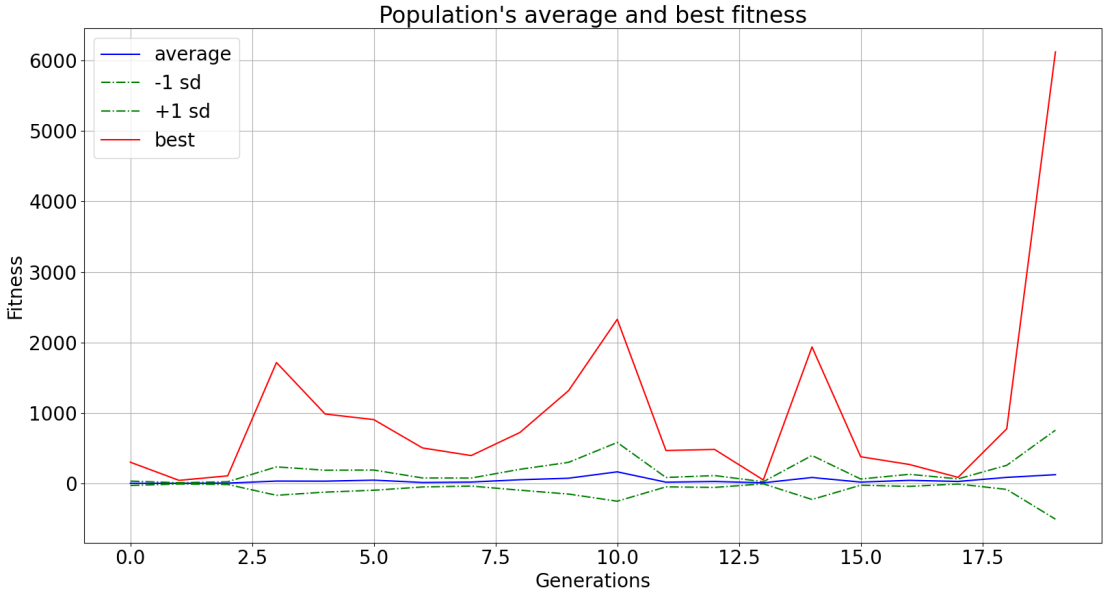
B. The Fitness function -

We will be using the same fitness function like before. This is because the value of fitness given as reward doesn't really affect the outcome. Even if we make the punishments harsher, the algorithm will be forced to pick the once a set of birds whose fitness value will be only marginally better than the rest. It will be like choosing a less worse option among many.

C. Outputs -

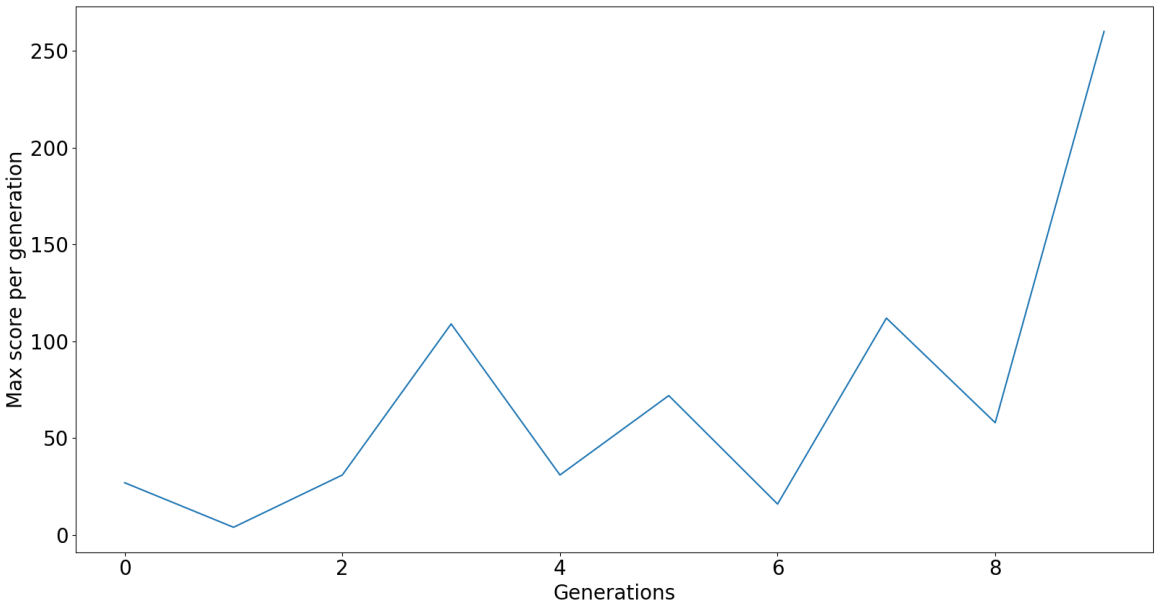
Now with the said corrections, we ran the simulation and obtained the following outputs after 20 generations for a population size of 100.

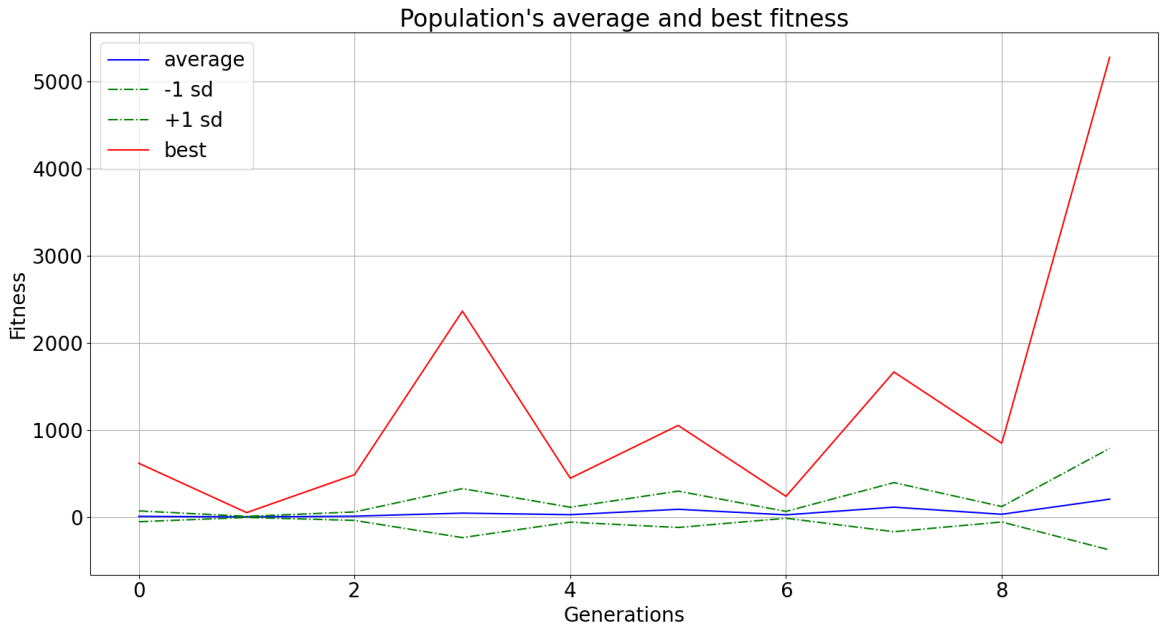




We can observe that the maximum scores of each generation has incremented several folds when compared to the earlier values. Also, the AI can now easily cover huge differences in height with ease.

Sometimes, the AI achieves a towering score within the first few generations, after which the programme terminates because the fitness value surpasses the fitness threshold (5000). In the output shown below, the AI manages to achieve a fitness value above 5000 on the 9th generation, thereby terminating the programme.





Conclusion

In this project, we discussed how to create an AI, using the NeuroEvolution of Augmenting Topologies algorithm, which can play the game “Flappy bird” indefinitely or for an infinite duration of time. This algorithm can be implemented using the N.E.A.T module which can be imported into python. In implementation of N.E.A.T, multiple AIs are produced which try to survive for the longest amount of time, which is done here by scoring more points. More the number of points, the more the fitness value. A certain percentile of AIs with top fitness are selected and are interbred to create the next population and the cycle goes on. This happens until the programme has completed the specified number of generations, or when the fitness value of the AI surpasses the fitness threshold. This results in the creation of an AI that can play the game indefinitely.

We can also get the same result by using deep Q learning, but the main disadvantage is that the training time required for it is far greater (almost 7 hours at times) than that required by N.E.A.T. Also due to the large size of the population, there is high probability for a good model to be created at the early stages, this is not possible in deep Q learning as only one model can exist at a time.

References

1. https://neat-python.readthedocs.io/en/latest/ (The official docs page).
2. https://towardsdatascience.com/how-do-we-teach-a-machine-to-program-itself-neat-learning-bb40c53a8aa6
3. https://www.pygame.org/docs/ (The official docs page).
4. https://github.com/CodeReclaimers/neat-python/blob/master/examples/xor/visualize.py (Code for visualize.py provided in the official docs-https://neat-python.readthedocs.io/en/latest/xor_example.html#visualizations)