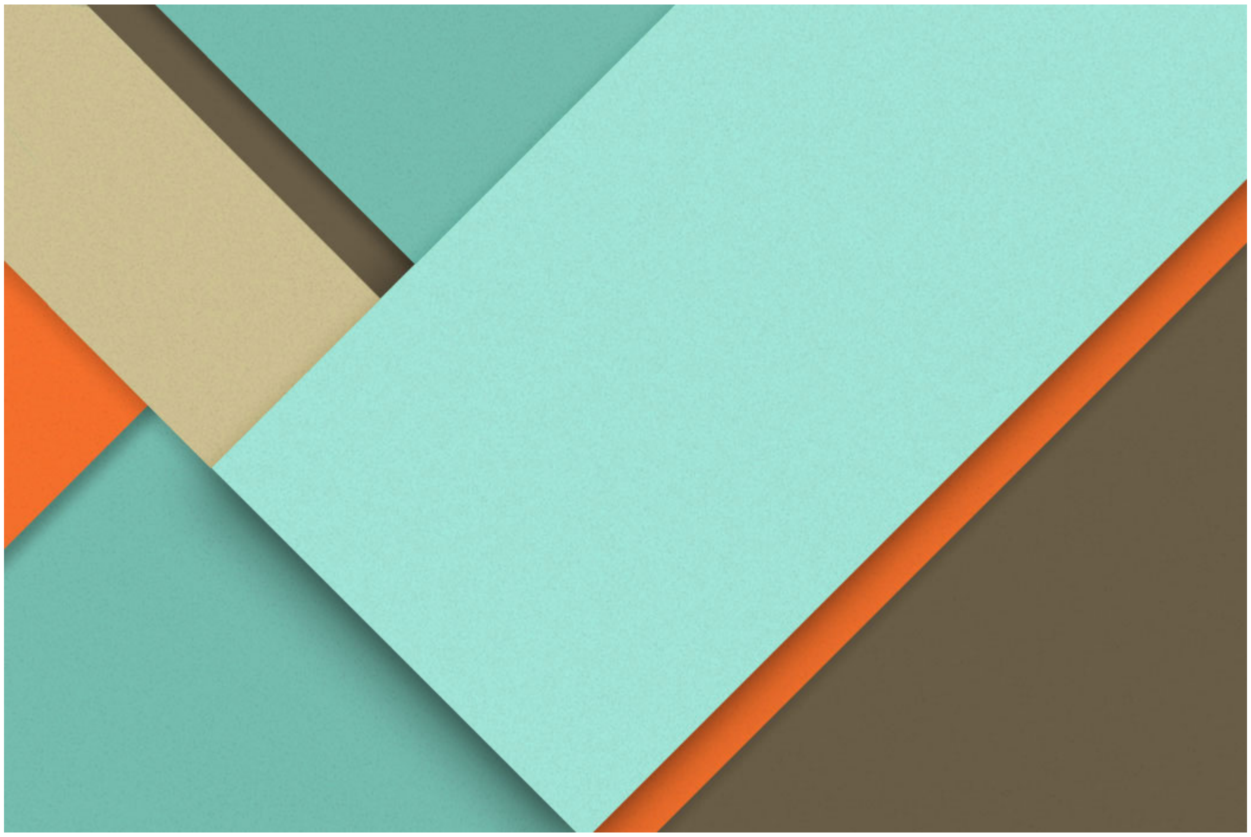


NeuroEvolution of Augmenting Topologies (NEAT)

03.24.2017

Lee, Suckgeun

Overview

NeuroEvolution of Augmenting Topologies (NEAT) is a reinforcement learning algorithm (RL) using Neural Network (NN) and Genetic Algorithm (GA) developed by Ken Stanley at 2002. NEAT first creates many NNs randomly, then use GA to find the optimal solution. It is known that NEAT performs well on simple tasks such as Pole Balancing problem; however, NEAT is also known for its poor performance at some complicated problems. In this report, I will implement NEAT from scratch, then test it on two relatively simple tasks and one complex task. For simple tasks, cart-pole problem [Barto83] and mountain-car problem [Moore90] are used; and for complex problem, Breakout from atari2600 game is used. The final goal of this report is to test if NEAT is capable of solving complex problem such as game playing tests. Every test will use OpenAI gym, and the NEAT algorithm I implemented is used, not other open source NEAT.

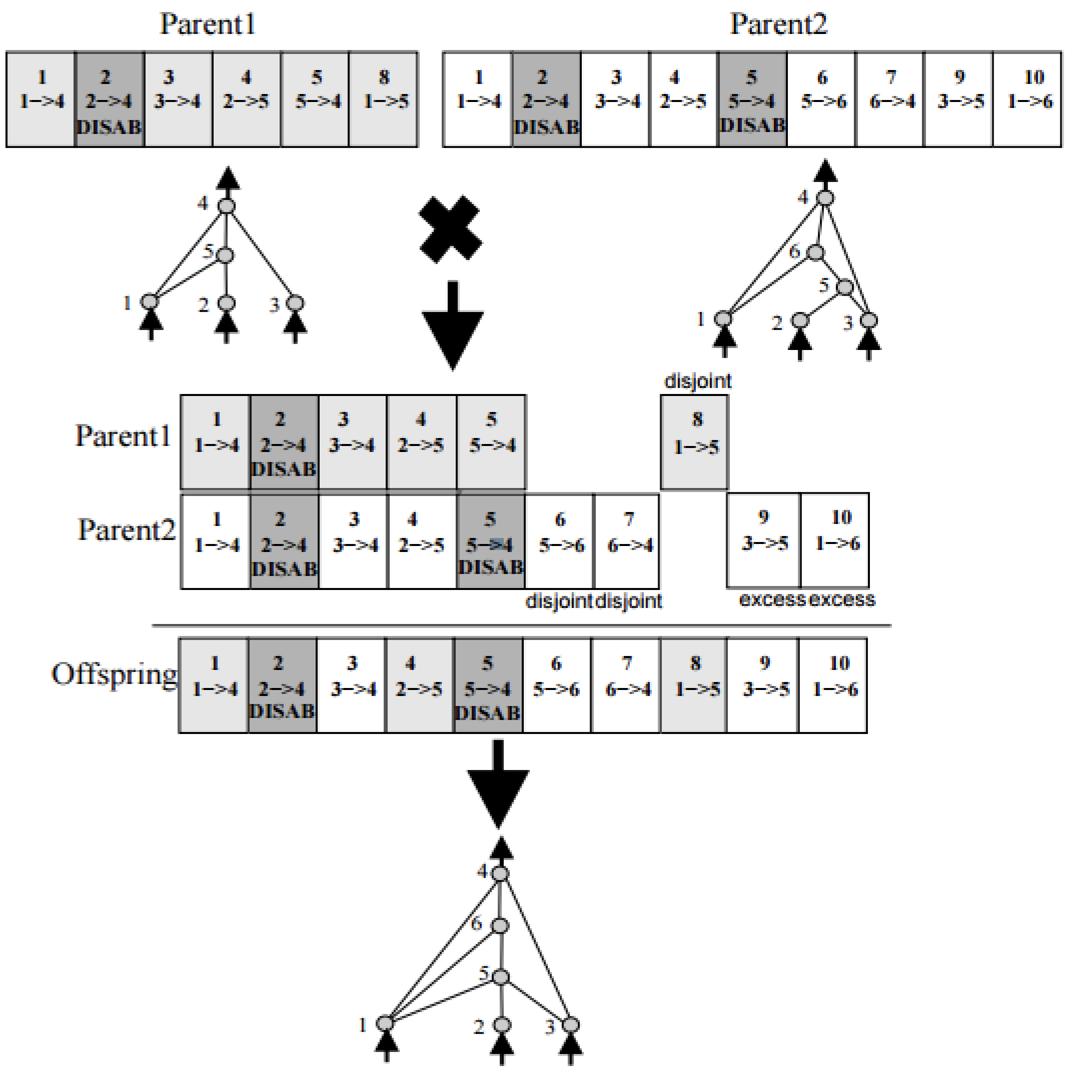
Introduction to NEAT algorithm

NEAT is developed by Ken Stanley at 2002. Unlike other popular NN algorithms, NEAT does not use backpropagation to find it's weights. Also, NEAT starts from very simple structure then slowly evolves itself to more complex structure. The evolving NNs is accomplished by using Genetic Algorithm(GA), and some brilliant tactics are introduced to effectively evolve both structure and weight of NN. The overview of NEAT algorithm is as follows.

1. Create multiple NNs with all input and output nodes are connected (first generation).
2. Run simulation for all NNs, then score(fitness) every NNs.
3. Adjust scores(fitness) of each NNs using “explicit fitness sharing(explained later)”
4. Drops inferior NNs
5. Crossover NNs (Creating next generation)
6. Mutate weights, connections, and nodes of next generation to change network structure.
7. Go to 2. until desired result is acquired.

The above list looks like just another regular GA algorithm, but what differentiates NEAT from other NeuroEvolution (NE) algorithms and GAs is how NEAT creates and manages its next generation. Since NEAT changes both weight and network structures, it is hard to crossover(making a child using two parents) with parents with two totally different structures. Also, when it changes its number of nodes and connections by mutation(changing structure with some probability), there is high possibility that the mutated new NN performs poorly, thus fails to survive against the old and more tuned NNs. To solve the problems, NEAT introduces "Historical Marking" and "Speciation".

When two NNs with different structures crossover, "historical marking" is used to check where and how to crossover. The NEAT paper introduces this process as follows.



<from NEAT paper>

Figure 4: Matching up genomes for different network topologies using innovation numbers. Although Parent 1 and Parent 2 look different, their innovation numbers (shown at the top of each gene) tell us which genes match up with which. Even without any topological analysis, a new structure that combines the overlapping parts of the two parents as well as their different parts can be created. Matching genes are inherited randomly, whereas disjoint genes (those that do not match in the middle) and excess genes (those that do not match in the end) are inherited

from the more fit parent. In this case, equal fitnesses are assumed, so the disjoint and excess genes are also inherited randomly. The disabled genes may become enabled again in future generations: there's a preset chance that an inherited gene is disabled if it is disabled in either parent.

NEAT also introduces "Speciation" to protect newly created structure. When NEAT mutates its network structures, there is high possibility that new structures are premature compared to the already tuned structures. To protect new structures and encourage to evolve NNs, NEAT groups NNs into different species using "explicit fitness sharing." Each NNs are evaluated its structure and assigned to a similar species. NNs compete each other within the species such that it gives new structure enough time to tune itself and survive. Details of explicit fitness sharing is explained well on NEAT paper.

By using these "historical marking" and "speciation", NEAT effectively changes its structure throughout the generation.

Parameters of NEAT

NEAT has quite a lot of parameters. Below is the explanation of each parameters.

1. n_input = number of inputs of each neural networks
2. n_output = number of outputs of each neural networks
3. n_nn = number of total neural networks to generate
4. activ_func = activation function to use. (default: sigmoid function)
5. bias = bias node. NEAT has only one bias node, and it always outputs same number.
6. c1, c2, c3, cmp_thr = parameters for "explicit fitness sharing"
7. drop_rate = when creating next generation, drop the poor performing NNs with this rate. (default: 80%)
8. weight_mutate_rate = how much weight should be changed when mutating weight
9. n_champion_keeping = decide how many best performing NNs to keep unchanged to the next generation (default: 10)
10. pc = probability of crossover. For this probability, crossover is executed. Otherwise, NN is just copied to next generation. (default: 75%)
11. pc_interspecies = probability of inter species crossover. (default: 0.1%)
12. pm_node_small = probability of node mutation with small species. (default: 3%)
13. pm_node_big = probability of node mutation with big species. (default: 30%)
14. pm_connect_small = probability of connection mutation with small species. (default:5%)
15. pm_connect_big = probability of connection mutation with big species. (default:70%)
16. pm_weight = probability of weight mutation (default: 80%)
17. pm_weight_ramdom = probability of randomly mutating weight (default: 10%)
18. pm_disable = probability of disabling a connection (default: 40%)

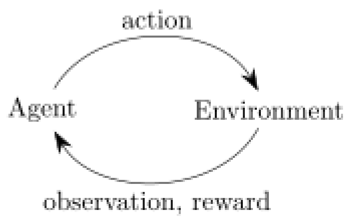
19. pm_enable = probability of enabling connection (default: 20%)
20. weight_max = maximum weight
21. weight_min = minimum weight

Among these parameters, number of input, number of output, number of NNs, bias, c1, c2, c3, weight mutation rate, weight max, and weight min will be adjusted for each test. Other parameters will use default values. The default values are from either NEAT paper or experiments.

OpenAI environment

In this report, OpenAI gym will be used as a test environment.

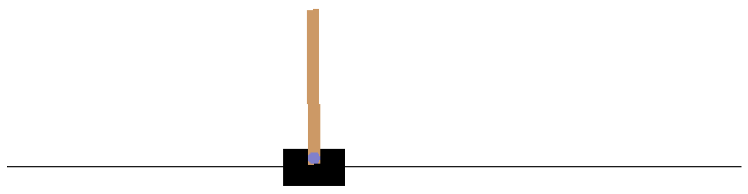
OpenAI uses the classic "agent-environment loop". Each timestep, the agent chooses an action, and the environment returns an observation and a reward.



<https://gym.openai.com/docs>

When game starts, environment returns observation and reward to the agent. The agents get the information, then choose appropriate action, then returns the action to the environment. This loop continues until desired reward is acquired.

Cart-pole test



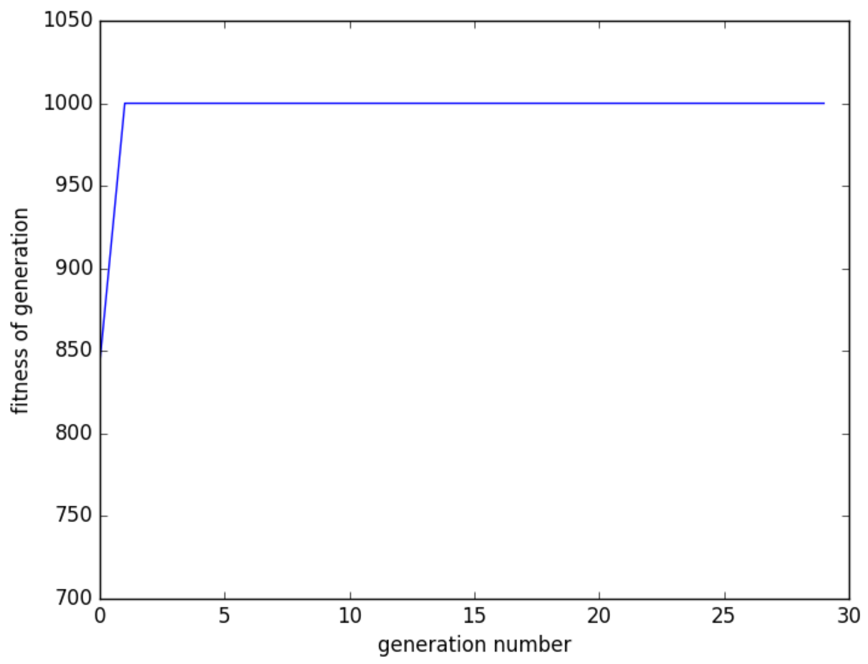
Here, I will test my implementation of NEAT using cart-pole problem. Cart-pole problem has a cart that moves along the horizontal axis, and a pole is attached to it. At every time step, one can observe its position (x), velocity (x_dot), angle (theta), and angular velocity (theta_dot). At any state, there are two actions the cart can take; go right or left. When the pole loses its balance, or the cart is off the screen, game finishes. Reward is given at each time step, so the agent gains bigger rewards if it can balance the pole longer. Cart-pole defines "solving" problem as getting average reward of 195.0 over 100 consecutive trials.

The parameters used for this test are as follows.

1. number of inputs = 4
2. number of outputs = 2
3. number of total NNs = 150
4. c1 =2, c2 = 2, c3 = 1
5. bias = 1
6. weight_max = 3
7. weight_min = -3
8. Weight_mutate_rate = 0.005

c1, c2, and c3 are chosen such that speciation creates sufficient species, and weight max, min, and mutate rate are chosen by experiments.

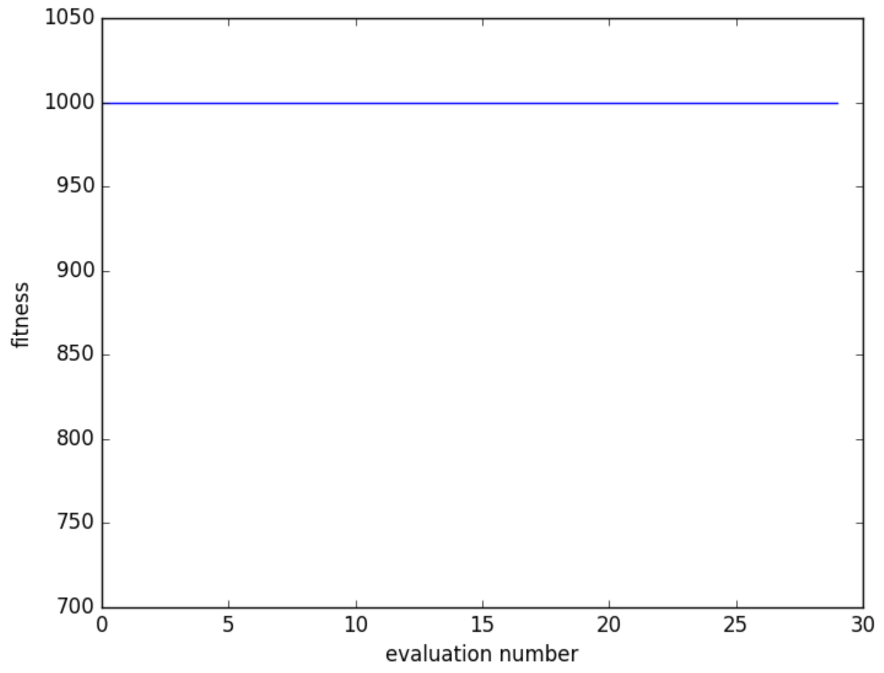
Total 30 generations are generated as test, and each generation's score is taken from its best individual NN. The maximum score is capped to 1000. Below is the test result.



<fitness of each generation>

As shown in the graph, NEAT founds the right NNs at the second generation. Since this test is easy to solve, it is expected NEAT to perform extremely well.

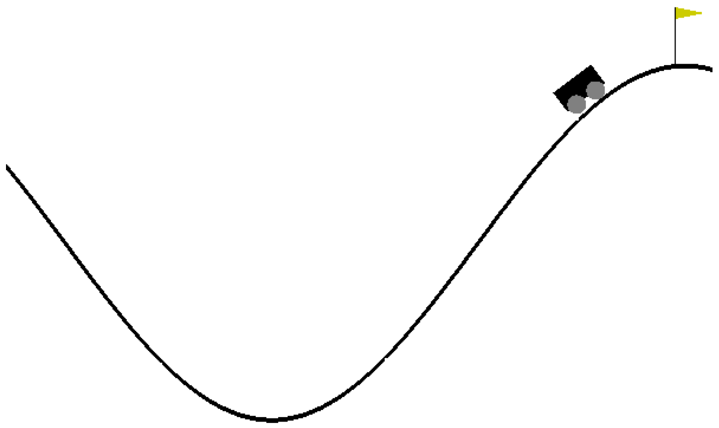
At the end of test, one NN having highest fitness is chosen. The NN will be run 100 times, and the result is as follows.



< evaluating the training result>

As the graph shows, the NN scores perfect(1000) every time, so the average reward is 1000. Since the condition for solving problem is getting average reward of 195 over 100 consecutive trials, it is safe to say that NEAT has solved the cart-pole problem.

Mountain-car test



Here, I will test my implementation of NEAT using mountain-car problem. Wikipedia explains Mountain-car as follows.

> Mountain Car, a standard testing domain in reinforcement learning, is a problem in which an under-powered car must drive up a steep hill. Since gravity is stronger than the car's engine, even at full throttle, the car cannot simply accelerate up the steep slope. The car is situated in a valley and must learn to leverage potential energy by driving up the opposite hill before the car is able to make it to the goal at the top of the rightmost hill. The domain has been used as a test bed in various reinforcement learning papers.

At every time step, one can observe the car's position (x, y) and the velocity of the car (v_x, v_y). At any state, there are two actions the car can take; go right or go left. At every time step, negative reward is given, and when the car reaches at the top of the mountain where the flag is at, the game ends. So the goal of the agent is to get to the flag as soon as possible not to get the negative reward. Mountain-car defines "solving" as getting average reward of -110.0 over 100 consecutive trials.

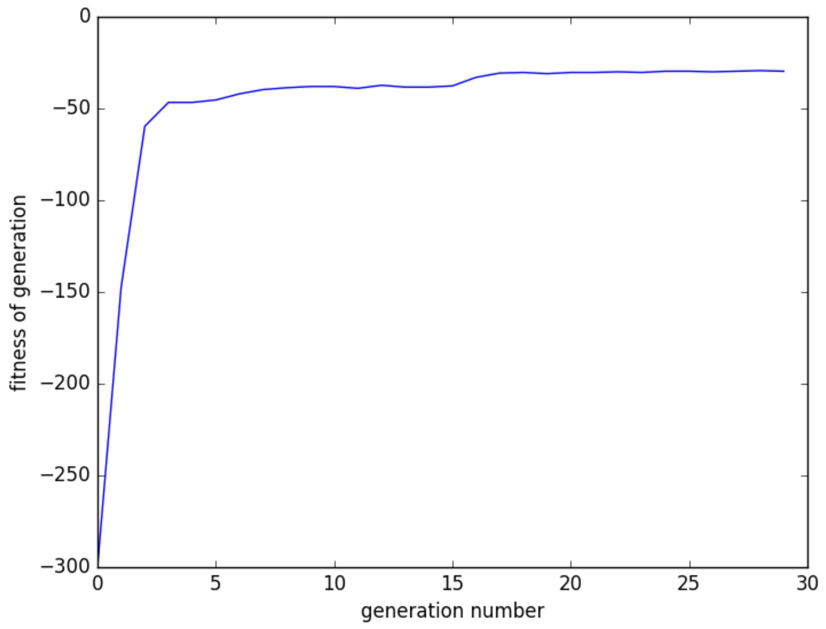
The parameters used for this test are as follows.

1. Number of inputs = 4
2. Number of outputs = 2
3. Number of total NNs = 150
4. c1 =2, c2 = 2, c3 = 1
5. bias = 1
6. Weight_max = 10

7. Weight_min = -10
8. Weight_mutate_rate = 0.1

c1, c2, and c3 are chosen such that speciation creates sufficient species, and weight max, min, and mutate rate are chosen by experiments.

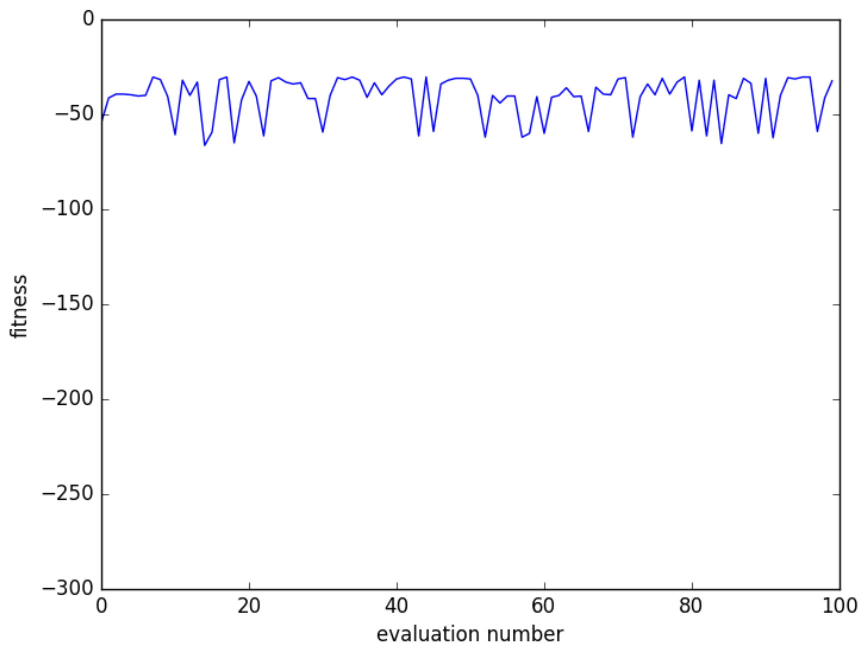
Total 30 generations are generated as test, and each generation's score is taken from its best individual NN. When the car fails to finish the task in 300 time steps, it finishes the trial and starts the next one. Below is the test result.



< fitness of each generation>

As shown in the graph, NEAT founds the right NNs very quickly. Before 5th generation, it already gets near -50 reward. Since this test is also easy to solve, it is expected NEAT to perform well.

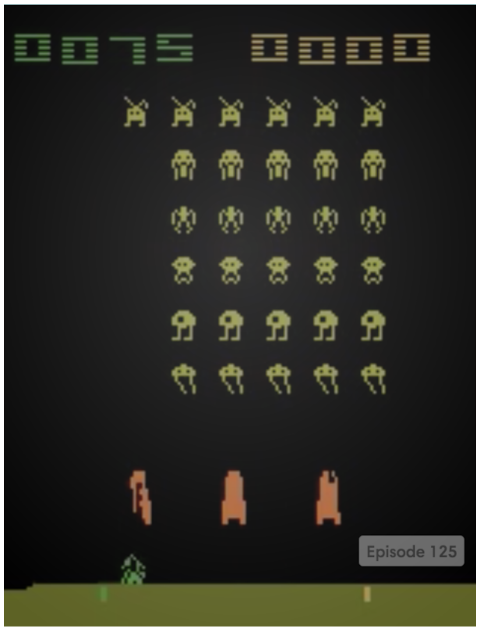
At the end of test, one NN having highest fitness is chosen. The NN will be run 100 times, and the result is as follows.



< evaluating the training result>

Mountain-car defines "solving" as getting average reward of -110.0 over 100 consecutive trials, and the graph is showing that every reward is above -100. So it is clear that NEAT solves the mountain-car problem.

SpaceInvaders (Atari 2600) test



<SpaceInvaders>

OpenAI describes the SpaceInvaders as follows.

> Maximize your score in the Atari 2600 game SpaceInvaders. In this environment, the observation is the RAM of the Atari machine, consisting of (only!) 128 bytes. Each action is repeatedly performed for a duration of kk frames, where kk is uniformly sampled from {2,3,4}.

On this test, the classic game called SpaceInvaders from Atari2600 in OpenAI gym will be used as testing complex environment. The goal of SpaceInvaders is to destroy as many invaders as possible while avoiding the enemy fires. As input data, RAM of the Atari machine will be used. About outputs, there are total 6 possible actions; do nothing, fire missile, move to left, move to right, move to left while firing a missile, and move to right while firing a missile. Destroying one invader gives 15 score. Unlike previous two tests, this test will not have “solving” defined, but the final score will be compared with random actions.

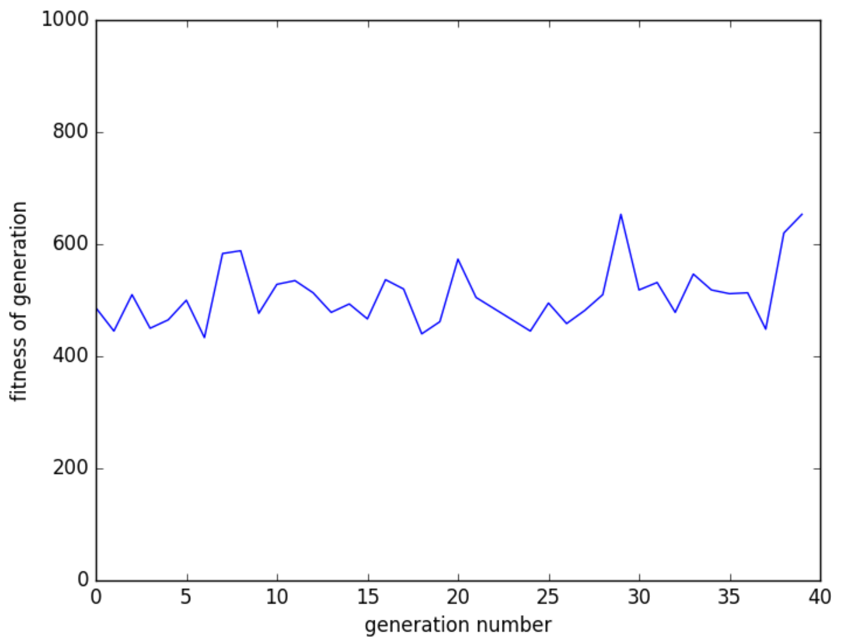
The parameters used for this test are as follows.

1. Number of inputs = 128
2. Number of outputs = 6

3. Number of total NNs = 150
4. c1 =300, c2 = 300, c3 = 150
5. Bias = 1
6. Weight_max = 3
7. Weight_min = -3
8. Weight_mutate_rate = 0.1

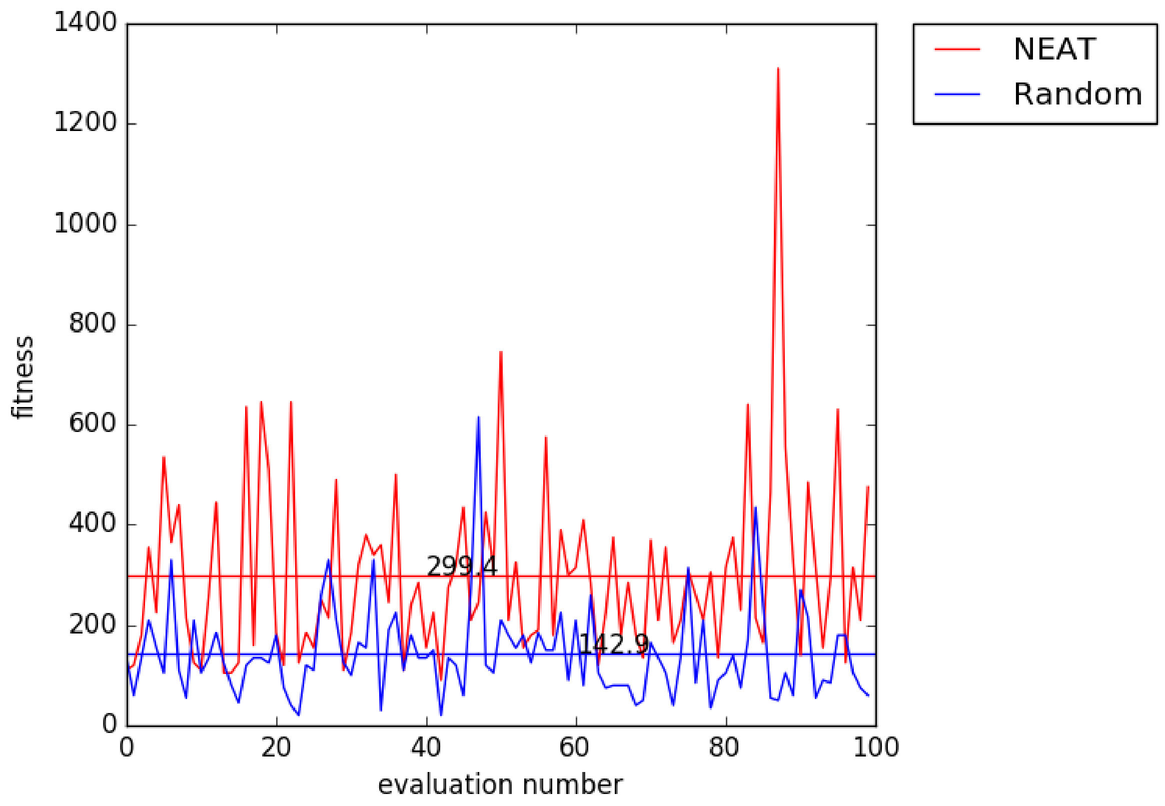
c1, c2, and c3 are chosen such that speciation creates sufficient species, and weight max, min, and mutate rate are chosen by experiments.

Total 40 generations are generated as test, and each generation's score is taken from its best individual NN. Below is the test result.



As shown in the graph, through the 40 generation, there is no significant performance improvement. Note that the recorded fitness is the best performed fitness, thus the actual fitness should be much lower.

The best scored NN is recorded and ran 100 consecutive trials. Below graph shows comparison between NEAT agent and random action gameplay.



The NEAT agent is getting average of 299.4 scores, and the random action game play is getting 142.9 scores. NEAT is learning to play the game, but the performance is not so great. OpenAI has a scoreboard, where one can upload their agent's score. Mean of the score on the scoreboard are above 400, so NEAT is performing poorly compared to other algorithms. Since the scoreboard does not explain what algorithms they are using, it is impossible to directly compare the results.

Conclusion

I have implemented NEAT from scratch and tested with two simple tests and one complex test. NEAT has performed well on simple tests, but it has performed poorly on complex task. I could not find any literature explaining why NEAT is performing poor. Because of lack of computing power, it was impossible to extensively test NEAT on complex problem; so further investigation is required.

References

Kenneth O. Stanley, et al, (2002), Evolving Neural Networks through Augmenting Topologies, http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

David Beasley, et al, (1993), An Overview of Genetic Algorithms :Part 1, Fundamentals, University Computing, 1993, 15(2) 58-69.

Matthew Hausknecht, et al, (2014) A Neuroevolution Approach to General Atari Game Playing