Creating a Traffic Merging Behavior Using NeuroEvolution of Augmenting Topologies

Yikai Wang and Ben Schreiber

December 15, 2015

Abstract

One of the main goals in developing an autonomous vehicle is programming the action of merging into the traffic lane from an entrance ramp. We seek to create such a behavior through the use of NeuroEvolution of Augmenting Topologies (NEAT) by evolving an agent over many generations to maximize a certain prescribed fitness function, which encourages a smooth merging behavior without crashing. Our experiment environment is simulated, and the agent starts from a fixed position on the entrance ramp and tries to merge into a separate lane with a constant flow of traffic. After evolving on many generations where the frequency of traffic on the highway is constant, say k, we have created an agent that can merge into the traffic lane seamlessly at the frequency of k, but only at k. Moreover, by training the agent on multiple environments where the frequency of the traffic varies within a single generation of NEAT, we have created an agent that can merge onto the highway given any frequency of traffic, as long as it is constant throughout the run and light enough so that the agent can physically fit in. There are, however, still limitations to this merging behavior, and we will point to some potential future research areas.

1 Introduction

Road traffic accidents are one of the leading causes of death in the world, and are often due to human error. Researchers have made significant progress towards autonomous vehicles, and one of the biggest challenges is to create successful merging behaviors without crashing into any other vehicle or causing any accident. For our project we seek to create a machine learning agent that can merge onto the highway from a halted position. Machine learning agents improve overtime as the experiment progresses, as opposed to hard-coded agents that always behave the same given the same environment. We controlled our agent with an artificial neural network (ANN), and used an evolutionary computation technique called NEAT to evolve our neural network.

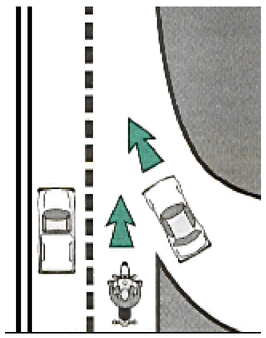

Figure 1: Merge visualization

1.1 Background of Neural Network and NEAT

To understand how we use NEAT to evolve desired behaviors, it is important to know a few facts about artificial neural networks. A neural network takes in numerical inputs from a given environment, substitutes these inputs into a formula, and then outputs a classification or behavior. The formula can be visualized such that inputs form a row or layer of nodes and the nodes each connect to nodes on the next layer through an edge with a varying numerical weight. Each node in the next layer will sum the activation coming from each node it is connected to in the previous layer multiplied by its edge weight and and passes this sum through an activation function (this explanation is recursive, one can think of the base case as when input layer takes in numerical values from the problem). After continuing this process for as little as one layer to any number of intermediary layers, eventually it reaches the final output layer which can be interpreted as a classification or behavior.

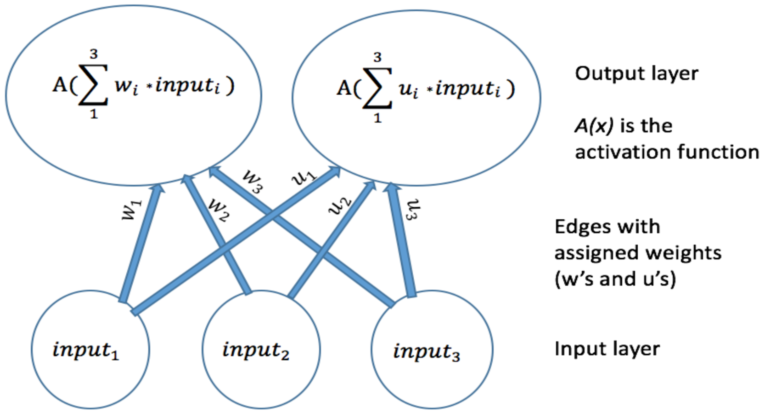
$A(\sum_1^3 w_i * input_i)$
$A(\sum_1^3 u_i * input_i)$
Output layer
A(x) is the activation function
w_1
w_2
w_3
u_1
u_2
u_3
Edges with assigned weights (w's and u's)
$input_1$
$input_2$
$input_3$
Input layer

Figure 2: A Simple Neural Network Topology Without any Hidden Layer

A natural question at this point would be "How does one configure the network so that it makes the right choices?" The answer to this question is highly problem specific in which there are many things to consider: how many layers of nodes there should be, how many nodes there should be in a middle layer, how the weights of the edges should be determined, etc. The number of configurations of neural networks is infinite, and our objective is to search through the space of them to find one that satisfies the given task.

We search for an optimal network by using NeuroEvolution of Augmenting Topologies (NEAT), a method created by Stanley and Miikkulainen[2] to find the optimal neural networks through the use of evolutionary computation. NEAT starts out with a simple neural network, and gradually complexifies by adding new nodes and connections. Weights are adjusted according to the feedback the network gets, and bad topologies are abandoned after several generations. This allows NEAT to explore increasingly more complex topologies, while maintaining efficiency. Each network is modeled as a phenotype consisting of node genes and connection genes. Node genes each have a number label and a label indicating whether they are input, hidden, or output nodes. Connection genes contain a mapping of two nodes, weight of that mapping, innovation numbers based on when they were created, and a field that indicates whether the given edge has been disabled. NEAT usually starts with a population of randomly generated networks each with just an input layer that is directly connected to an output layer. Upon creating the next generation, the best individuals (individuals with the highest fitness values) in the current population are paired with those from the same species (i.e. similar graph structures), and mated to create new individuals. During this mating process mutations may occur at random to the offspring such as changes of weights between existing edges, additions of edges between previously unconnected nodes, and additions of new nodes which complexify the network.

1.2 Related Work

Neural networks have been used to produce autonomous vehicles in various studies. In the research by Mitsuru Tanaka [3], artificial neural networks were used to develop self-driving behaviors. The ANN Model successfully generated car-following behaviors that outperformed the traditional GM Model, which was hand-coded and developed by the General Motors Company. However, Tanaka chose a neural network arbitrarily with two hidden layers, each of which has 5 nodes. We would argue that NEAT can potentially produce better results because it explores different topologies.

Another application of Neural Networks in highway driving scenarios is the work done by Huval et. al. in [1]. In this paper the authors examined some of the problems with classic computer vision techniques in recognizing environmental objects and characteristics such as cars or lane markers. Instead of mitigating these problems with the use of intensive road-modeling and and special case handling, they employed the use of a convolutional neural network (CNN) to make these classifications. These networks are a special case of feed forward multilayered neural networks in that individual neurons within the structure are responsible for overlapping regions in the input images. Although these networks had proven successful at many imaging tasks, they had not proven to be accurate under real time conditions. The authors determined that in order to keep up with the pace of real-life driving, the networks would have to operate at speed of 10HZ while processing images of 640X480 pixels. The results were staggering — after training on 616 thousand image frames with lane annotations, their network was able to learn to identify cars at 60 meters away with $\approx 95\%$ accuracy while only using a single graphics card, and with speed of 44Hz. Our work is complementary to this study. Huval et al. trained a network for purely image classification tasks so that it can be fed into a behavioral determination system. Our system is in a simulated environment where inputs are assumed to be perfect, and neural networks are used to create optimal behaviors in an agent. In order for an autonomous vehicle to succeed on the road, both of these mechanisms need to function synchronously.

2 Experiments

We conducted our experiments in a simulated environment (Figure 3). We have two lanes of traffic separated by an orange line, and our agent starts from a still position on the lower lane. Its goal is to merge onto the upper lane, which has a constant flow of traffic. The agent merges successfully when it reaches the right end of the upper lane, which is marked with a star in Figure 3. In our environment, the agent is modelled as a blue circle with a radius of 40 pixels, and the cars that represent traffic are modelled as red circles with the same radius. We gave the agent 1500 time steps to merge, which is sufficient because the vast majority of the agents either reach the end goal or crash well before the time steps have past. For the sake of simplicity, we configured the cars in traffic to always move at a constant speed of 10 pixels per time step.

We ran three different experiments within the environment described above. In the first experiment, the agent attempts to merge into traffic with a new car emerging after every 50 time steps, allowing plenty of space for the agent to fit in between traffic cars. In the second experiment, a new car emerges in the traffic lane after every 20 time steps, allowing it much less space between cars to merge. In our final experiment, every agent is trained on four different traffic frequencies per run of NEAT, where a new car emerges after 20, 30, 40 and 50 time steps respectively. An agent's fitness after the four rounds is the average score it received for each round.

2.1 NEAT Configuration

The outputs from our NEAT network must not be limited to positive values (as we will explain later), and therefore we used Tanh function as our activation function, because its outputs range from -1 to 1. The Sigmoid function, which is often used as the activation function in NEAT, only allows outputs from 0 to 1, and is therefore not suitable for our experiment.

We have initialized the starting population of networks to have no hidden nodes, and the probabilities for adding connections and nodes are 0.1 and 0.05, respectively. In each of the experiments described above, we ran 100 generations, and there were 100 individuals in every population.

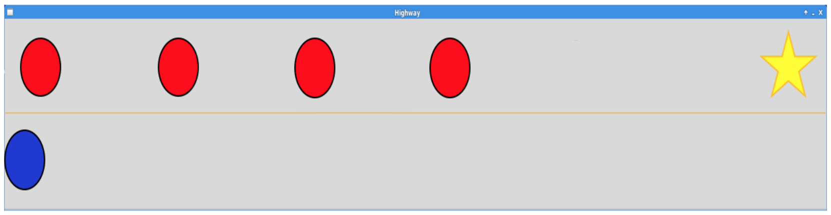

Figure 3: Simulated Environment

2.2 Inputs and Outputs

The NEAT network has 7 inputs. The network will be fed its agent's current speed, its position which includes x and y components, its heading, the speed of cars in the traffic lane, and the horizontal distance between the agent and the nearest car in front, as well as behind (see Figure 4). We believe these inputs are a good abstraction of how human drivers perceive the traffic while merging.

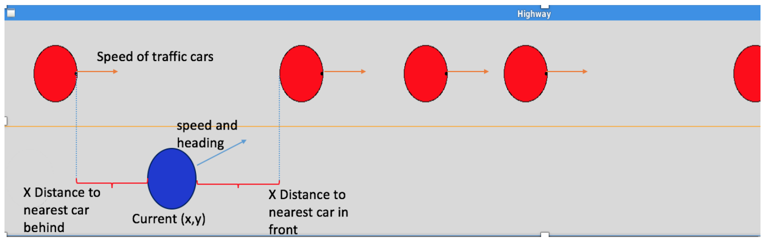
Highway
Speed of traffic cars
speed and heading
X Distance to nearest car behind
Current (x,y)
X Distance to nearest car in front

Figure 4: Visualization of NEAT Inputs

The outputs of this network will be acceleration [-1,1] and rotation [-1,1], just as a real vehicle has an accelerator, break, and wheel. An acceleration output of 1 corresponds to a maximum acceleration of increasing 5 pixels in one time step, and -1 corresponds to a maximum deceleration of velocity decreasing 5 pixels from one time step to the next. We capped the velocity of the agent at 10 to simulate the speed limit of cars, and also because it is the speed of the cars on the traffic lane. If the agent can reach a speed faster than the rest of cars, it can potentially hit the end goal very easily just by beating the others, which defeats the purpose of our experiment. We also do not allow the agent to go backwards as well; in this case, we keep the velocity above or equal to 0 at all times. A rotation of 1 corresponds to a hard rotation to the right, which is 5 degrees, and -1 corresponds to a 5-degree rotation to the left. Any output in between will cause a proportional acceleration or rotation in the agent.

2.3 Fitness Function

Our fitness function is on a scale of [0,1] where 0 is the worst and 1 is the best fitness. We agreed that successful agents were ones who were able to merge onto the upper lane of traffic and made it to the top right corner. These agents would be assigned a preliminary fitness of 1. If an agent finished in the upper lane, but did not make it to the top right corner, its preliminary fitness value would be a function of its x position; if it finishes in the lower lane, we assign it an arbitrarily small fitness value. Moreover, if the agent oscillates back and forth between the traffic and entrance ramp we subtract off 0.1 from its score for each time it does this (after the initial one, which is necessary for merging). If the agent crashes with one of the other cars or the wall, we halve its preliminary fitness (a function of its x position at the time of the crash) to penalize it (see Figure 5). Finally to differentiate behaviors which complete the goal quickly from those which don't, we subtract off 0.0001 points for every time step taken during the run. For example if an agent makes it to the goal in two time steps its fitness would be 0.9998, if an agent takes three time steps its fitness would be 0.9997. Thus, the agent who took less time has higher fitness. The following is the pseudocode for this fitness function:

```
Run 1500 time steps (if it finishes early we stop)
**if** an agent has met the goal **then**
    fitness ← 1
**if** it hasn't met the goal and is in the upper lane **then**
    fitness ← function(final x position)
**if** not in the upper lane **then**
    assign an arbitrarily small fitness value
**if** crashed during the runs **then**
    run stops and fitness ← fitness ÷ 2
fitness ← fitness −(numCrossovers-1)*0.1
fitness ← fitness − timeSteps*0.0001
```

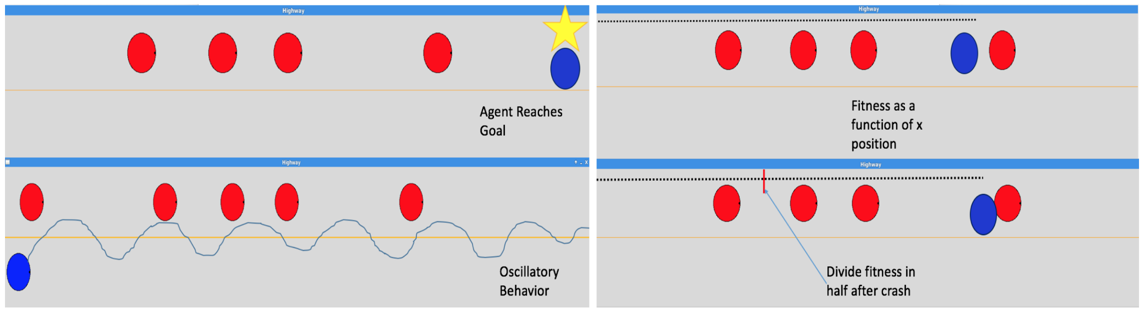
Highway
Agent Reaches Goal
Fitness as a function of x position
Oscillatory Behavior
Divide fitness in half after crash

Figure 5: Some Factors in Evaluating Fitness

3 Results

Our hypothesis is that NEAT will be successful in training a robot to merge onto the highway successfully. In our first experiment we wanted to evaluate this hypothesis on the most basic conditions, where traffic cars would come at a rate of 1 car per 50 time steps of the program (see Figure 6). This allows plenty of space between cars for the agent to merge.

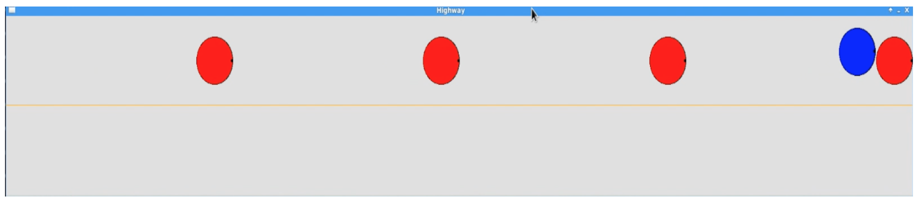

Figure 6: Traffic at a Frequency of 1 Car per 50 Time steps (Demonstrations available on YouTube channel, musicAlgo.)

The results from this training were successful— after evolving for 100 generations, NEAT produced agents that could solve the task by reaching the goal state of the upper right-hand corner of the environment in a reasonable amount of time. In Figure 7, we see that NEAT finds an optimal behavior at approximately generation 27, and finds one that finishes the task even faster at around generation 34. Average fitness increases steadily throughout the evolution starting at approximately 0, and ending at 0.2.

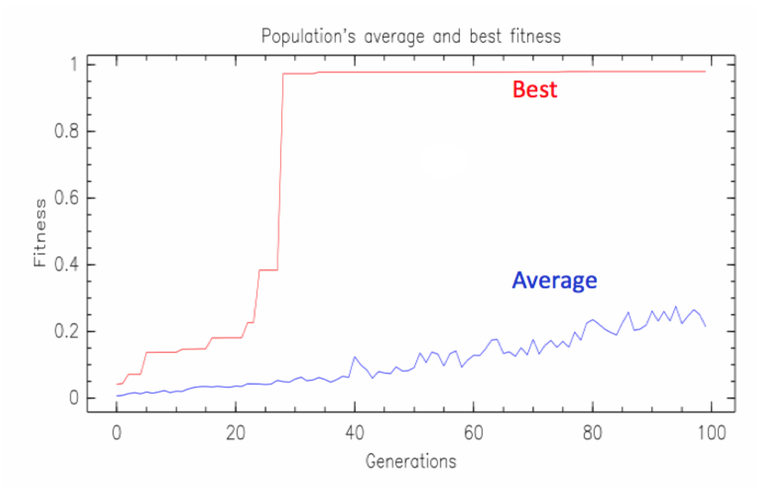

Figure 7: Average and Best Fitness per Generation of NEAT at Frequency of 1 car per 50 time steps.

While these results were promising, they were on an environment in which the frequency of traffic was very light. (i.e. there was much space in between cars for the merger to fit itself in.) To test whether NEAT can successfully evolve an agent that merges in a more challenging environment, we increased the traffic rate to 1 car per 20 time steps (see Figure 8). We determined empirically that this was almost the maximum rate at which the merger could possibly fit in the flow of traffic. If it were much heavier, there simply would not be any window to merge.

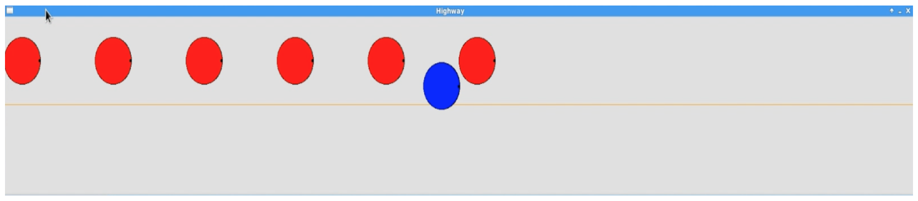

Figure 8: Traffic at a Frequency of 1 Car per 20 Time steps

Results for this experiment were similar in that average fitness increased at a steady trend, and the best fitness was found early, this time at generation 15 as shown in Figure 9. This demonstrates the ability of NEAT to evolve an agent to solve a difficult task.

Since this behavior could merge under such intense conditions we wanted to see if it could adjust to less difficult environments. In particular, we wanted to see if the best agent could perform well in an environment where the frequency of traffic was constant, but different from the level on which it was evolved. To do this we tested the agent on traffic frequencies ranging from 1 new car per 10 time steps to 1 new car per 80 time steps (that is, 1 test per integer value in between 10 and 80). In Figure 10 we can see that the evolved agent still performed significantly better than the best individual in the randomly spawned initial generation, but did not perform well overall. The behavior was optimal only under the conditions on which it was evolved

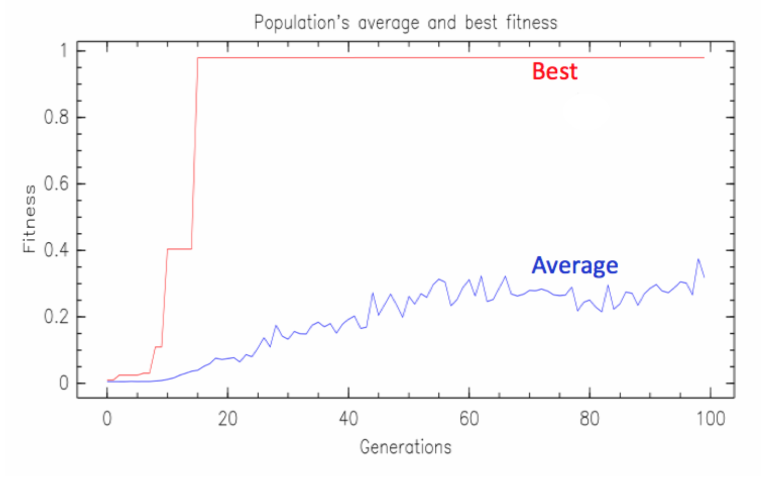

Figure 9: Average and Best Fitness per Generation of NEAT at Frequency of 1 car per 20 time steps

(even though they were the most difficult). We also applied the same test to the agent evolved on a rate of 1 car per 50 time steps, and it too did not merge well under frequencies equal or less intense.

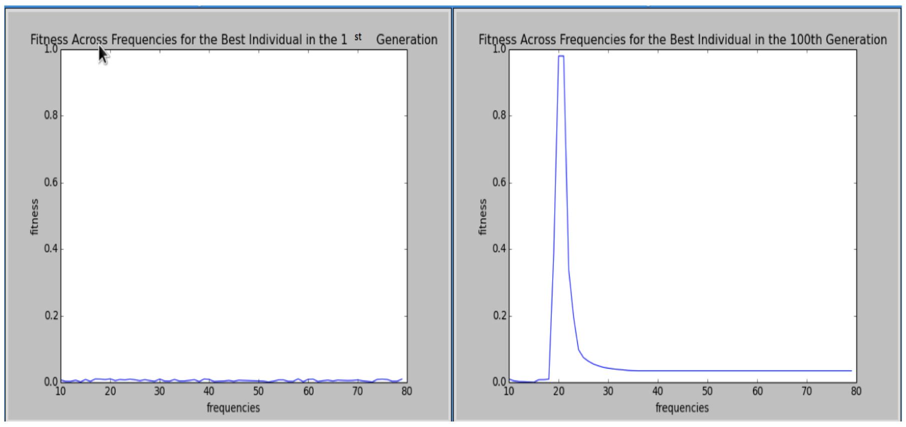

Figure 10: Fitness Across Frequencies after Evolving on Frequency of 1 Car per 20 time steps

We have found that NEAT can produce agents that merge excellently under the exact same environment in which they were evolved, but could not adjust to new environments even if the tasks are easier. In our third experiment we sought to create an agent that could be adaptable on a single variable in its environment, and we chose this variable to be the frequency of the traffic. Precisely, this means that given any constant frequency of traffic less intense than $\approx$ one car per 20 time steps, our agent would be able to merge successfully.

To do this, we tested each agent on multiple environments with frequencies of 1 car per 20, 30, 40, and 50 time steps. Fitness was calculated as usual for each environment and averaged. For instance if an agent received a scores of 0, 0.1, 0.2, and 0.3, its fitness would be $\frac{0.6}{4}$. We hoped that at a minimum, this agent would be able to perform well on the environments in which it was trained. Indeed, after evolving for 100 generations NEAT was able to find a behavior that performed optimally on all the environments in which it was evolved. Figure 11 demonstrates that by generation 46, NEAT had already discovered an agent that can reach the end goal in all four environments.

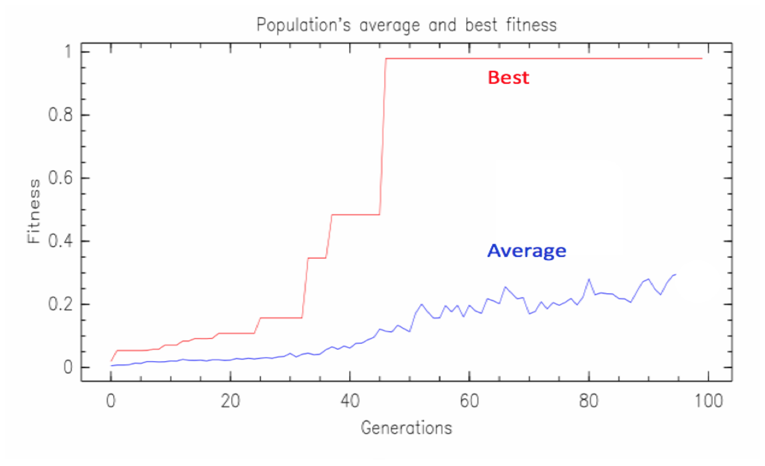

Figure 11: Best and Average Frequency per Generation on Multiple Frequencies

In fact, the best agents produced in this experiment can merge in any environment with a frequency ranging from 1 car per 18 time steps to 1 car per 80 time steps, as demonstrated by Figure 12. This means that given any integer frequency between 18 and 80, the best agents trained on multiple rounds will merge successfully. No agent from the first two experiments exhibits this level of adaptability. If you are interested in seeing results, please visit https://www.youtube.com/watch?v=s6MxJsHTzKM or visit Ben's musicAlgo channel on YouTube.

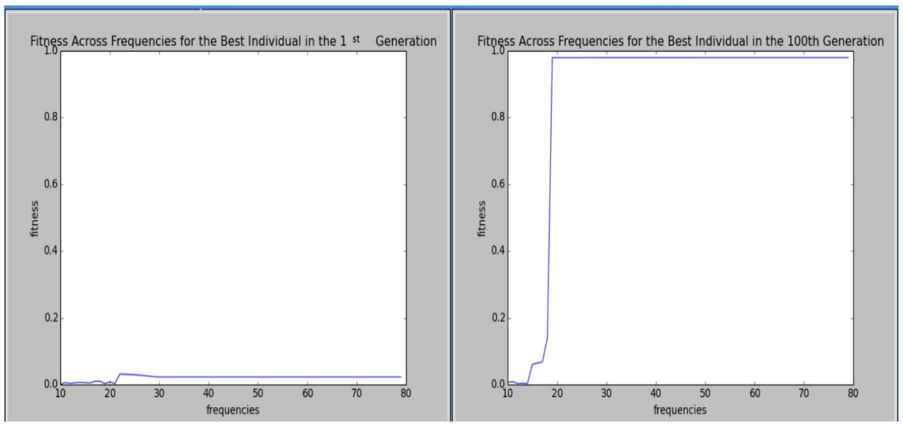

Figure 12: Fitness Across Frequencies after Evolving on Multiple Frequencies

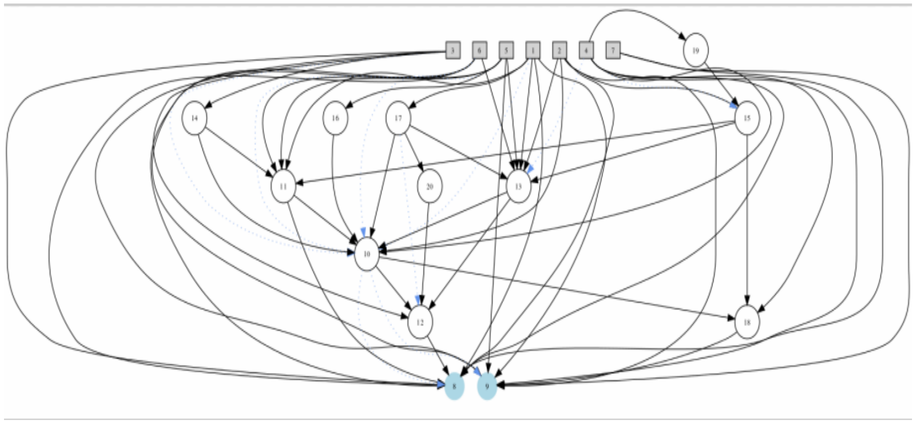

Figure 13: Topology of Best Individual Behvavior

4 Discussion and Future Work

Through this research we have demonstrated the power of NEAT to evolve behaviors that meet a given task. After evolving on 1 car per 50 time steps, NEAT found a driver that can merge on to the highway, and after evolving on 1 car per 20 time steps it also found a successful behavior. Upon seeing the ability of NEAT to produce individuals that succeeded in the same environments on which they were trained, we sought to alter our approach so that NEAT would find an agent that could succeed in environments where the frequency of traffic was different from that on which it was trained. By evolving on several environments per generation each with distinct traffic frequencies, NEAT was able to create an agent that can merge onto the highway at any constant frequency of traffic between 1 car per 18 time steps and 1 car per 80 time steps, and potentially beyond. With NEAT, we have managed to create an adaptable agent.

The results are promising, but are a long way from achieving the level of adaptability needed to succeed on the actual roads of today. In real life the frequency of traffic varies, cars on the traffic lane accelerate and decelerate, and the length of the entrance ramp on which the agent can merge may vary. Therefore, one immediate thing to consider when adding to the functionality of this agent, is incorporating adaptability on other variables, such as changes of traffic frequency within an environment, the starting position of the agent, and varying speeds of cars on the traffic lane. To add this functionality we will likely need to add more rounds of tests per generation of evolution since there are more variables to consider. Even though our current evolutions take 1-2 minutes to run, adding additional variables could significantly increase the amount of computational time needed. One way this could be implemented without adding too much computational burden would be to take the minimum and maximum value for every dimension and test under every possible combination. This would yield 2^n rounds of tests per generation for adaptability on n variables. Despite the aforementioned drawbacks, the development of autonomous vehicles may benefit from this evolutionary computation approach.

References

[1] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Royce Cheng-Yue, Fernando Mujica, Adam Coates, et al. An empirical evaluation of deep learning on highway driving. *arXiv preprint arXiv:1504.01716*, 2015.

[2] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research 21*, 2004.

[3] Mitsuru Tanaka. Development of various artificial neural network car-following models with converted data sets by a self-organization neural network. *Proceedings of the Eastern Asia Society for Transportation Studies*, 2013.