The applicability of the NEAT Neuroevolution algorithm on driving fast and wrecking opponents.

Emil Bigaj
Martin Frisk
Rahul Setty

May 30, 2017

Abstract

In this report the Neuroevolution method, Neuroevolution of Augmenting Topologies (NEAT) generated artificial neural networks for the purpose of controlling an agent in the open source racing game Trophy. It is shown that NEAT can be used to generate highly effective racers able to navigate different tracks and beating human players. Furthermore agents are trained that generalize to race tracks unseen in training, while maintaining super-human performance.

Contents

1 Introduction

A popular application of artificial neural networks (ANN) is to train them to play video games [11]. Depending on the nature of the game different training regimes are more or less applicable. A problem that arises when training an ANN to successfully play a game by mapping game states to actions, is that the evaluation of the outcome is not easily described as a differentiable function. This means that gradient descent is not available as a training option. Training an ANN to play a video game may be further complicated if the design of a suitable network topology is non trivial.

Neuroevolution (NE) provides a solution to these challenges by using evolutionary algorithms to build and train ANNs. These methods only require a user to be able to evaluate the performance of an ANN to be applicable. In this sense they are highly flexible to problem settings and can be used for supervised, unsupervised and reinforcement learning [13]. This makes NE a popular training scheme for video game playing.

A NE scheme proposed by Stanley and Miikkulainen [18] called *Neuro-Evolution of Augmenting Topologies* (NEAT) has been particularly popular due to several of its properties. In this study, NEAT is applied to the open source racing game Trophy [10], a vehicular combat racing game with features such as shooting, mine laying etc. The scope of this study is to see what aspects of the game NEAT is able to learn, and to compare how different NEAT/Trophy configurations perform in this regard.

2 Neuroevolution

In Neuroevolution the training of ANNs (adjusting link weights and/or the topology) is driven by evolutionary algorithms. The process produces a population of potential solutions which evolve improved performance over a number of generations. Each generation begins with a population of organisms, where each organism contains a genotype which encodes an ANN, the phenotype. The population is evaluated by applying each of its organisms ANN to solve the problem of interest and allocates to them a numeric fitness score based on performance. Offspring are then produced from the existing population by crossing over and mutating the genomes of the most successful organisms, the organisms with greatest fitness score. This process forms the population of the next generation. The procedure continues until some stopping criteria such as satisfactory performance on the problem is reached.

NE has repeatedly been show to be a successful training tool for various types of video game playing agents [13]. NE offers several features which make it an attractive option in this setting.

1. NE solutions have provided the best performance in a variety of game settings. In numerous games NE solutions performed better than alternative artificial intelligence approaches, see [2] [20] [5] for examples.

2. NE is applicable to a wide range of problems. The only pre-requisite for training with NE is the ability to allocate a numeric measure for the performance of an ANN. This allows NE to be successful when training by gradient descent is infeasible. NE can be used for training in supervised, unsupervised and reinforcement learning settings [13].

3. NE has been shown to scale very well with increases of input space [13] [14], [12]. In the case of video game playing, the description of the game state as well as possible actions an agent may take can be of very high dimensionality.

4. Some NE techniques can automatically build a ANN topology. The ability of an algorithm to build an architecture for an ANNs is especially useful in the

case of video game playing where the complexity of decisions needed to be made by translating a game state into and set of actions requires a non trivial ANN architecture. Even where the user already has a working topology, NE may discover simpler solutions and variations.

2.1 Neuroevolution of Augmenting Topolgies

NE covers two types of algorithms. The first are those which only adjust the link weights of a given network topology and the second are those which not only adjust link weights but also adjust the network topology.

Of the approaches which only adjust link weights the methods, CoSyNE [6], ESP [7] and CMA-ES [9] have been shown to have superior performance compared to topology adjusting NE approaches in a variety of benchmarks [5]. However, their crucial drawback in the video game playing setting is that they require the user to defined the topology.

In the class of topology adjusting NE, Cellular Encoding [8], and Analog Genetic Encoding (AGE) [3] are competitors to NEAT. However NEAT has several properties which make it more attractive than its counterparts [18]. When compared to CE and AGE in the performance of solving the double pole balancing benchmark (a popular benchmark test for NE methods), NEAT has been found require less CPU time [5]. Further, unlike NEATs counterparts, NEAT begins with a minimal topology and augments it, resulting in simpler solutions [18]. For a complete review of NE methods see [4].

3 Relevant Work

Various forms of NEAT have been applied to interact with a broad range of video games in different roles [13]. These roles have covered, state/action evaluation, selection between playing strategies, modeling opponent strategy and direct action selection [13]. This study applies NEAT in direct action selection. NEAT has been used in this way to play GO [19], Keep Away Soccer [20], Battle Domain [15] and various Atari 2600 games [14]. For a full review see [13].

In the case of car racing, like this experiment, the most relevant work has been [2] where NEAT was used to train an agent in the racing game TORCS. However in this application the goal was to produce the best racer possible by combining NEAT evolved agents with preprogrammed directives. The aim of this experiment on the other hand is to produce a racing agent based exclusively on a NEAT trained ANN with no programmed instructions.

4 Neuroevolution of Augmenting Topologies

Like other NE methods, NEAT is driven by a genetic algorithm. However NEAT provides several innovations which solve some fundamental short comings of previous methods. (1) NEAT uses a gene encoding scheme which allows disparate topologies to cross over in a meaningful manner [18]. (2) NEAT maintains genetic diversity by protecting innovations to topology by speciation so they may become optimized and properly evaluated [18]. (3) NEAT finds simple solutions by beginning with a minimal topology and extending it in each generation by adding new nodes and links, so that simple solutions are found before complex ones [18].

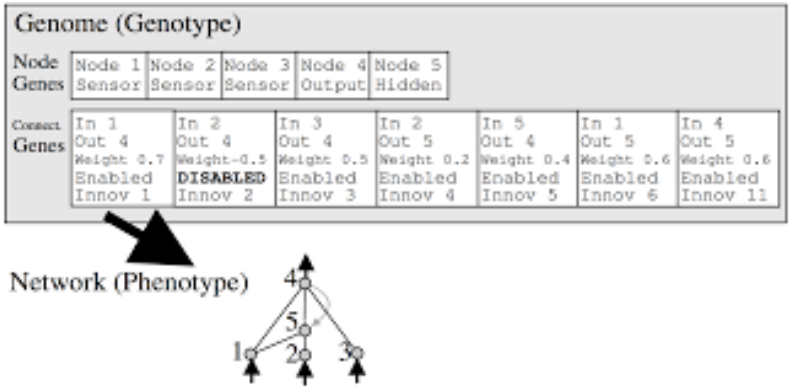

Figure 1: Genome encoding and representation [17]

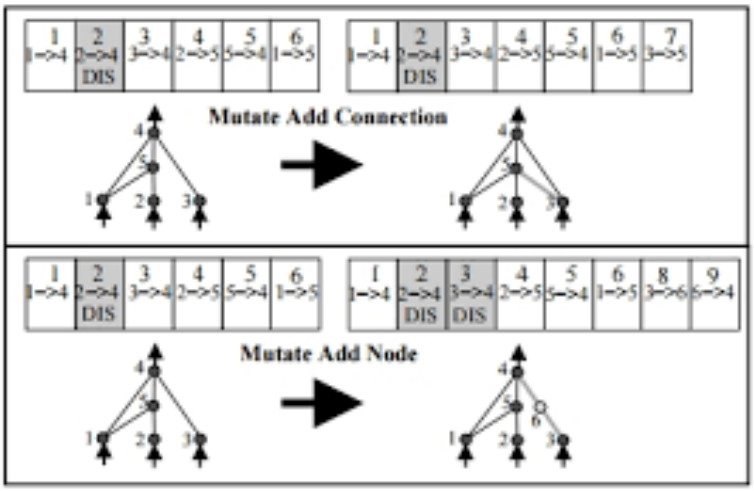

Figure 2: Mutation in NEAT [17]

4.1 Genome Encoding

To allow NEATs evolution of network topologies, a genetic encoding scheme which is expandable and dynamic is used [18]. Each genome consists of a collection of connection genes (fig. 1). Each gene describes a connection between two nodes by specifying an in-node, an out-node and a weight associated to the connection. In addition to these a gene also contains an enabled/disabled bit to allow a gene to be turned on or off and an innovation number which tracks the origin of the gene.

4.2 Mutation

NEAT allows three different mutation operations (fig. 2). Add link : A new connection gene is added where there was no connection between the nodes previously in the genome. Add node : A new node is added in between two already existing nodes. The old connection is disabled. Weight change: The weight of a node is perturbed by a small random amount. There is also a small chance that the weight is assigned a new random value, completely independent of the previous one.

4.3 Innovation numbers

Innovation numbers act as a historical marker for a gene. The function of these historical markers is to inform NEAT of the genetic origin of the gene. If two genes have the same innovation number it indicates that they encode the same topological

structure. Tracking the historical origin of a gene in this way means that if two genes have an identical innovation number, they must represent the same topological structure so that crossover between two genomes holding these genes produces a valid topology. Each time a new connection is added through mutation it is assigned an innovation numbers from a global counter.

4.4 Crossover

NEAT performs crossovers of connection weights and also topology (fig. 3). Crossover of connection weights occurs when the innovation numbers of genes match. In this case the connection weights of the fitter parent are adopted. If both parents have equal fitness, connection weights are randomly selected with equal probability. The offspring from topological crossover adopt the disjoint and excess genes of both parents. A disjoint gene is a gene present in only one parent, but where the innovation number of that gene is less than the greatest innovation number of the other parent, while an Excess gene is a gene with innovation number greater than the greatest innovation number of the other parent.

4.5 Speciation

NEAT splits the population of organisms into distinct groups known as species [18] (fig. 4). Species are constructed in a way such that organisms with similar topologies are allocated to the same species. During selection for breeding an organisms competitors are limited to the others of the same species. When a mutation occurs which augments the topology (adding a new node or link) it is possible that although in the long run this augmentation can improve the behavior of the ANN (after the new link weights are given time to be optimized), it may be the case that in the short run, a decrease in performance is experienced. By limiting competition between organisms to those of the same species these augmentations are given opportunity to be optimized (adjust link weights) so that their long run benefit can be realized. Without speciation genetic diversity is limited because selection for breeding will be biased towards the best organisms of whole population, discarding organisms with a potentially better topology but experiencing worse short term performance because of its lack of optimization.

In order to divide the population into different species where each species contains a set of similar genomes, NEAT uses a measure of genetic distance. The aim of this measure is to return high values when there is a large number of differences in topology and/or a large difference in average link weights of the matching parts of the topology. The genetic distance measure can be written as a linear combination of the number of excess(E) and disjoint(D) genes and the average weight difference between the matching genes, $\overline{W}$ [18]

$$\delta = \frac{c1E}{N} + \frac{c2D}{N} + c3\overline{W}$$

The coefficients c1, c2 and c3 are used to adjust the importance of three factors and N is the number of genes of the larger genome [18].

To assign the genome to a species, the distance measure δ is calculated between the genome and a randomly chosen genome from the species. If the distance measure δ is less than a threshold, the genome is placed in that species. In this way, NEAT makes sure that no genome is placed in more than one species [18].

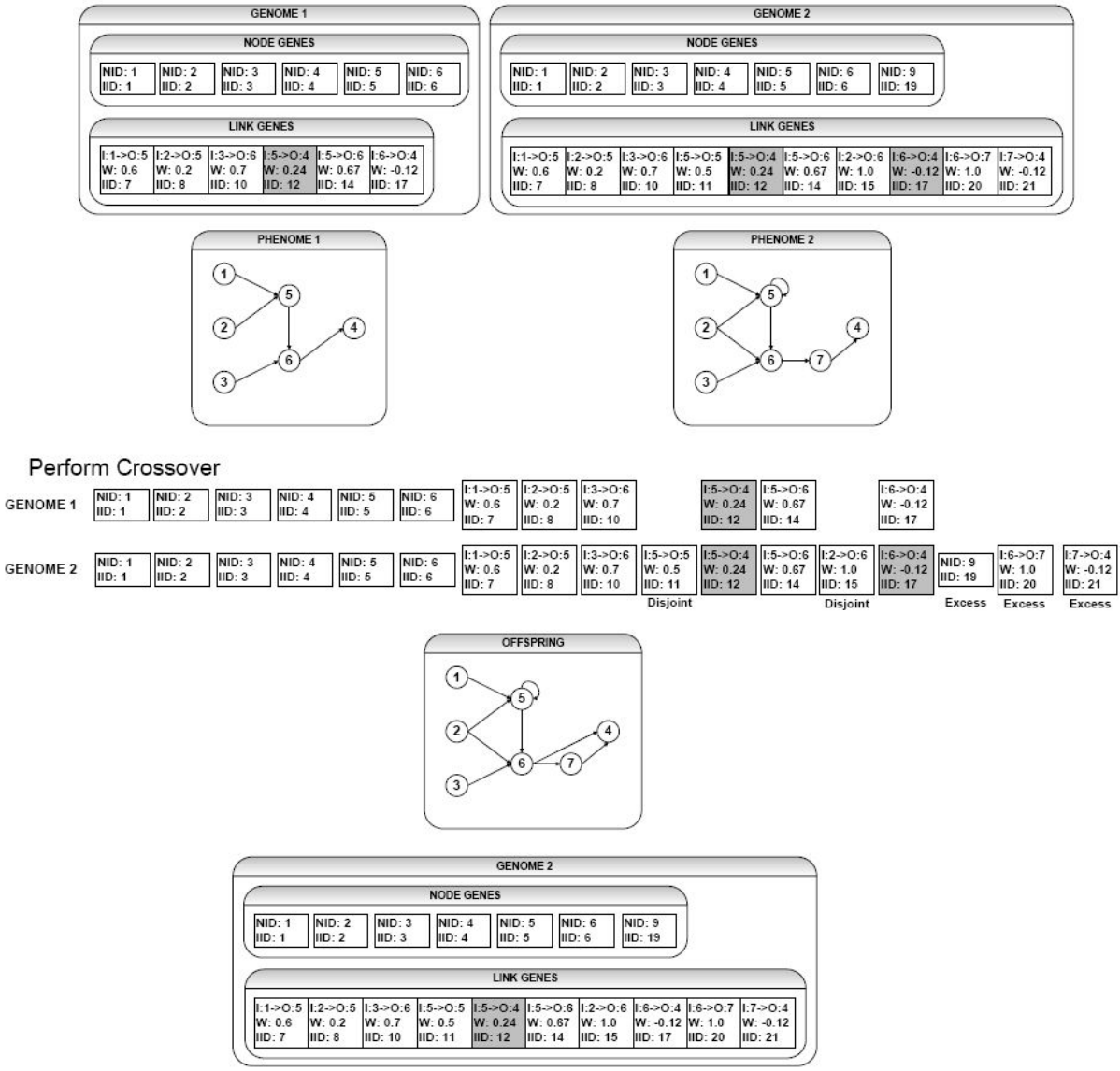

Figure 3: Crossover in NEAT [1]

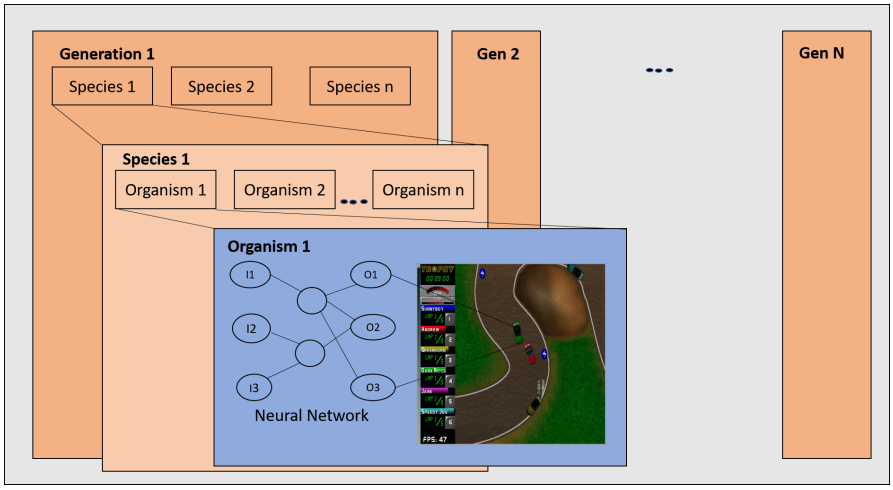

Figure 4: Speciation in NEAT

5 Neuro Evolving Racers

5.1 Trophy

Trophy is a vehicular combat racing game. Like a regular racing game, the ultimate goal is to be the first to complete the race, however in this game opponents are able to attack and destroy each other. Destroyed vehicles are disqualified from the race. Therefore a good racer would finish the track quickly while also causing damage to opponents while avoiding receiving damage from opponents. Vehicles are able to damage each other by colliding, shooting and dropping bombs in front of them.

Trohpy provides an interesting example of a game to be trained by NEAT because of the variety of behaviors required to be successful, namely finishing the race quickly, damaging opponents and avoiding damages to oneself. This study however has been limited to only consider racing as an individual.

5.2 NEAT C++

To apply NEAT to Trophy the NEAT C++ library [16] written by K. Stanley et al. for their original original paper [18] was used.

5.3 Driving by Neural Network

A race is broken up into discrete time steps. At each time step, each player provides a set of actions to be performed until the next time step. Each player is controlled by an agent by evaluating an ANN. This ANN takes as input the game state from the perspective of the agent and produces output which controls the agent until the next time step. This feedback loop is how the game is played out by the agent.

5.3.1 Inputs

The inputs are numbers extracted from the game, which represents information that an agent can base decisions on. As such it represents a very important design choice.

In [14], the authors used an input space consisting of raw pixel data, which is very general in the sense that it is applicable to any game. Evaluating the performance of several learning schemes they show, among other things, that NEAT may not be well suited for this type of input space. They did show however, that a variant of NEAT called Hyper-NEAT (which is beyond the scope of this study) could learn games using only pixel-data.

Speed Limit

0	15	15	15	7	7	7	7
0	15	15	15	15	7	7	7
0	15	15	15	15	7	7	7
0	0	15	15	15	15	7	7
0	0	15	15	15	15	7	7
0	0	0	15	15	15	0	0
0	0	0	15	15	15	0	0
15	15	15	15	15	15	0	0

Figure 5: Grid based game state

In light of the above, two schemes for representing a game state were considered. Both have the property that they are centered at the location of the agent, and rotated to align with it's direction forward.

The first approach is a grid based state space that measures at grid points in a square grid how passable the terrain is at a corresponding point in the game (fig. 5). For instance, a solid wall is represented as a 0, while the road is represented as 15. Values in between exist.

The second configuration is ray based and is designed to liken vision (fig. 6). A number of rays, evenly spread out across the field of view are shot out. Each ray extends until it hits an obstacle, and the distance it traveled is the corresponding input.

These approaches are not general in the sense that they rely on having data interpreted before it is presented to the agent, and if one were to apply this approach to another game, that interpretation would have to be adapted. One should also note that one can calculate at least a good approximation of the ray based game state from the information in the grid based one. This means that agents using the grid based game state could at least in principle learn the same tasks and perhaps more than the ones using the ray based one. However, it seems likely that the ray based game state represents information that is more easily applicable to a driving agent, and that it would simplify training.

To increase the complexity of the game, both configurations have the possibility to be extended to also show objects, such as cars and power-ups. The number of spatial inputs will then be doubled, so that each grid point (or direction) has two numbers. One representing the terrain, and one that denotes the presence of objects. Objects are coded as numbers, which are projected onto the closest grid point/angle.

5.3.2 Outputs

In total there are 8 different actions which can be performed during each time step. There are 4 actions which control player movement: accelerate, decelerate, turn left, turn right. The 4 other actions control: fire guns, drop bomb, use turbo and sound horn. The ANN contains 8 output nodes each corresponding to one of these actions. The output nodes all contain a sigmoid function which evaluates to a value between 0 and 1. After activating the ANN, output nodes producing values greater than 0.5 are considered activated, and the game executes corresponding actions.

In these initial experiments, the allowed outputs were restricted to be only the ones controlling the movement of an agent. Furthermore, interaction between cars was disabled, and thus the inputs are limited to show only the terrain.

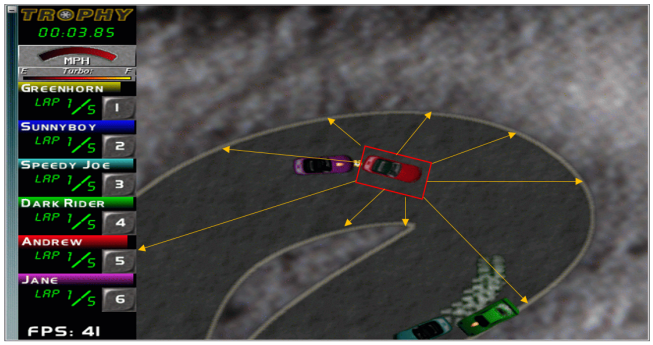

Figure 6: Ray based game state

6 Training

6.1 Evaluation

During training by NEAT each organism must be given a fitness value. This is computed by evaluating performance of racing. The population is divided into heats[1], which end either by all players dying or giving up, or by a maximum time limit being reached. At the end of the heat each participant's fitness is evaluated and assigned to the corresponding organism. For training purposes, the game engine was set up so that the heat does not end ever by a player completing some number of laps.

Many different fitness functions were tried, the most successful ones being

$$\phi_1 = k \cdot d$$
$$\phi_2 = k_1 \cdot (t + k_2) \cdot d$$

where t is time spent in the game, and d is the distance it traveled along the track, and $k, k_1, k2$ being constants. Due to the training setup described above, it follows that t is bounded explicitly by a constant, whereas d is only implicitly bounded by the rules of the game.

Time spent in game varies significantly because of strict rules forcing an agent to give up unless it fulfills certain conditions listed below:

1. The agent must activate at least one output at all times.
2. The agent may not fail to increase in d for a sustained duration.
3. The agent must not be slowed down by collisions too much.
4. The agent must always change it's position from one frame to the next.

The interplay between the fitness function and the rules for giving up are important to understand. ϕ_1 is very direct, as increasing the traveled distance in the direction of the track will per definition make you successful at playing Trophy. Having this as the ultimate goal will naturally subsume every other relevant sub-goal, such as surviving, driving fast and not taking a longer route than needed.

ϕ_2 seemingly incentives driving slowly, if for instance there is a difficult part of the map that the agent is unable to get through. This may be true to some extent, but it may not always be bad for learning as driving slowly into a difficult section could be a

[1]The division into heats has no impact on training in the current set up, but it is needed to facilitate future tests where cars can interact with one another.

Figure 7: Tracks - Zurich (left), Snake (middle), Rally (right)

good learning strategy. The key observation is that for all agents which do survive to the maximum time limit, ϕ_2 degenerates into $c \cdot \phi_1$ (for some proportionality constant c), so they do share the same long term goals.

In section (7) a comparison between the functions ϕ_1,ϕ_2 is given.

6.2 Tracks

Agents were trained on three different tracks (fig. 7). Separate experiments were performed by training on individual tracks as well as all three tracks in combination. In the case of training on individual tracks, the agents were tested on the other two tracks to check for ability to generalize.

6.3 Random Start Locations

Considering what happens in the first generations of NEAT offers some insight into a way possibly reduce training time. In a normal Trophy race, each car begins at the point of track designated as the start of the race. If each training race begins from this location NEAT will preferentially breed those racers which can navigate the start of the race and potentially discard racers which are poor at the start of the race but could be superior in other parts. If we take the track Snake as an example we notice that the race begins with a straight and is followed by a right turn. Suppose two racers exist in the first generation which both drive forward during a straight but one racer is able to turn left while another is able to turn right. The racer which can turn right will receive a greater fitness score than the racer which has learned to turn left, although when taking the whole track into consideration, both racers are equally good. As a result NEAT will allocate more offspring to the racer turning right and possibly discard the racer turning left. To evaluate these racers more fairly random start positions were used. Each racer was placed in 6 random start locations and the average fitness scored of these were used.

7 Experimental Results

7.1 Progress

Improvements in performance of the best organism generally showed long periods where improvements in fitness stagnated (fig. 8). The reason for this is that NEAT would regularly spend many generations failing at a particularly difficult part of a track, such as a sharp turn. Once such a turn was successfully navigated the fitness score of the best organism would jump to a new high. In addition to this, the jumps in best fitness indicate the generation number where the agent learned to avoid colliding with the outsides of the track, a behavior which is severely penalized (section. 6.1).

7.2 Tracks

Agent training was largely effected by track design. A track with greater variety of turns (left and right of different angles) is more difficult to learn, compared to a track

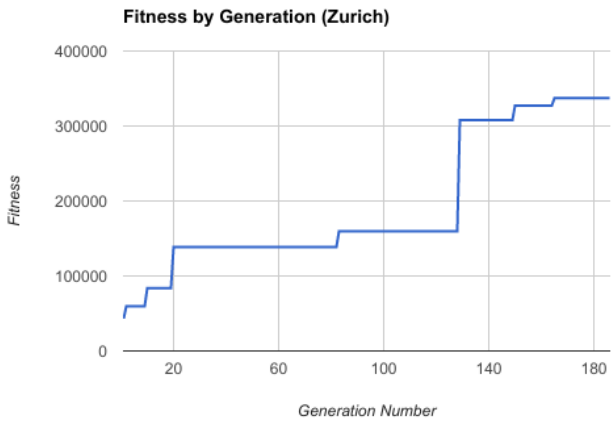

Figure 8: Best Fitness by Generation Number

with smaller variety of turns. This is evident from the finding that an ANN with only three links was sufficient in complexity to successfully navigate through the entire track Zurich (albeit inefficiently) but not for example Rally. Further, efficient driving in Zurich, that is, completing the track in a near perfect manner required only 20 generations of training whereas 79 generations were required to even complete Rally (fig. 7).

One factor that makes maps hard for this setup that is easily overlooked is the fact that agents does not have any memory. A human player will surely memorize the surprising turn of Snake and play accordingly, whereas an NN-agent only has a very primitive representation of it in it's inputs. The ray-based game in particular may struggle with such situations as a very sharp turn will not necessarily "look" different to the agent, compared to a not so sharp bend.

Below is a table of how many generations of training on the different maps (training taking place on one map at a time), were required to reach the objectives of completing a lap[2] and becoming super-human[3]. Numbers are averages of the training sessions that did manage to reach the goals. The first two tables are for the ray-based game state, the third being for the grid-based one.

It appears by these results that ϕ_2 provides quicker learning. It is likely because it provides incentives that encourages safe driving early on, while later adding speed. ϕ_1 often resulted in very reckless driving strategies in the first few generations that do not constitute a good foundation for future learning.

Ray-based game state fitness function ϕ_2:

Map	#generations to complete a lap	#generations to become superhuman
Zurich	6	20
Snake	110	-
Rally	46	79

Ray-based game state fitness function ϕ_1:

Map	#generations to complete a lap	#generations to become superhuman
Zurich	17	80
Snake	-	-
Rally	89	-

[2]This is the number of generations after which one NN-agent managed to go one lap, however poor the driving.

[3]Here the experimenters performance is taken as representatives of humanity.

Grid-based game state fitness function ϕ_2:

Map	#generations to complete a lap	#generations to become superhuman
Zurich	37	122
Snake	130	-
Rally	-	-

7.3 Game State

Agents trained on the grid based input space were unable to complete Rally because of a confusing feature of the tracks design; there are several turns where different parts of the track lie next to each other separated by a thin wall. When approaching the turn the agent sees the track beyond the separation and drives towards it, failing to understand that the separating wall between tracks is an obstacle. The agent is aware that the wall exists but it's behavior is not sophisticated enough to understand that track beyond it is unreachable. In theory, NE should be able to learn to how to handle such a track feature but with limited training that was not the case.

The ray-based game state overcomes this, by presenting the information to the agent in a more understandable form: it simply does not get the information about what is on the other side of an obstacle - only the distance to it. As seen in experiments, this representation has two other substantial advantages:

- Training is faster because input space is smaller. To present the information of terrain features with sufficient precision to facilitate learning, our experiments found that an 11×11 grid was needed. The ray-based representation functioned well for as low as 9 sensors.
- The representation lends itself to formulating generalizable rules. This is because states that require similar action are more likely to look similar on every relevant sensor than they would in the grid-based game state.

7.4 Generalization

Generalization between the tracks was proven difficult. For instance, due to the non-obvious difference between Zurich and Rally - in that the side of the road on rally is still quite passable, but still registers as impeding movement - agents trained on Rally will learn that it is good to cut corners. This behavior is punished heavily on Zurich, where the side of the road registers similarly, but it causes a much greater slow down.

As Snake is such a confusing track, any training regime that involves it usually provide no benefit for learning other maps.

Zurich and Rally do share some features, and that shows in generalization experiments. Training for very large number of generations (100 or more) usually yields agents that have learned both some general driving rules and some map specific heuristics. This is a form of over-fitting, and does not generalize well. If one stops training before this happens, agents often generalize to one of the other tracks.

A fast, but off-track avoiding driving style from Rally does very well on Zurich and vice versa. Indeed, the best performing agents on Rally were trained on Zurich. The reason for this seems to be that the simplicity of Zurich makes it very good for learning. On Zurich, a driver which drives very simply performs well, and there is not really much point to learn any specialized maneuvers.

Agents that specialize in this type of driving do well on Rally because this track allows for simple driving schemes - but only if they are sufficiently good. If training on rally, the many twists and turns tempts the agents to learn various specialized movement patterns for short term benefit, that actually do not provide the best foundation for future learning.

There may be some hyper-parameters for NEAT that prevents this from happening, such as balancing the probabilities for the different mutation types (more focus on adding and balancing weights on links, and less on new nodes for instance) but due to the vast parameter space of NEAT, it is considered beyond the scope of this study.

8 Future Work

This study has been limited to single player racing in Trophy. However, Trophy is a multilayer game where each players actions effect those of others. As such a huge scope of investigation into the viability of NEAT for training an agent capable of reacting to opponents is an opportunity for further work. Specifically several questions remain unanswered.

An effective way of modeling the location and behavior of opponents remains unexplored. As found in this work a grid based approach may provide a simple solution to this but may again to prove less useful when compared to solution which provides more meaningful information.

Adding more complex game playing options such as combat could be implemented in two ways. NEAT could be used to produce a single ANN capable of both driving and attacking opponents. Investigation here will also surely raise questions about how the increase in dimensionality due to the extra information supplied to an ANN would effect training time. Another approach worth considering would be based on two or more ANNs where each ANN is trained for a specific action say driving or shooting. In this case the objectives of racing and destroying opponents would be treated independent of each other. This would however constitute a substantial supplementation of the NEAT library.

In the case of a single ANN responsible for shooting and driving a new fitness function would have to be defined for training which rewards good behavior in both goals. On the other hand, the alternate approach where several ANNs are used independently of each other, would allow for a simpler individual fitness functions.

Another direction for further studies could be to add some form of memory of previous game states as inputs to the agent. In the present state, agents act only on the information that they perceive in the same moment as they act. This makes the agents easily confused, as for instance they have no way of knowing whether they turned around or not when they crash.

Also the many parameters and settings of NEAT could be studied in much greater detail, to determine their impact on learning, as there are surely more optimized parameters for learning to play Trophy still to be found.

9 Conclusions

In this work NEAT was applied to generate ANNs to control racing cars in the video game Trophy. The investigation here was limited to cars racing without opponents. Two different fitness functions, and two game state-representations were evaluated.

On two tracks out of the three studied, the authors were utterly defeated by the agents trained. Moreover agents trained on one map could play another, and still defeat the mere humans to which they were compared.

The NEAT algorithm preformed quite well in training from scratch, with little information about the task it was set to solve, using only very primitive representations of the game state.

References

[1] A. Adrift. (2017) Competitive coevolution through evolutionary complexification. [Online]. Available: http://www.automatonsadrift.com/neat/

[2] L. Cardamone, D. Loiacono, and P. L. Lanzi, "Evolving competitive car controllers for racing games with neuroevolution," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 1179–1186.

[3] P. Dürr, C. Mattiussi, and D. Floreano, "Neuroevolution with analog genetic encoding," in *Parallel Problem Solving from Nature-PPSN iX*. Springer, 2006, pp. 671–680.

[4] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[5] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Efficient non-linear control through neuroevolution," in *European Conference on Machine Learning*. Springer, 2006, pp. 654–662.

[6] ——, "Efficient non-linear control through neuroevolution," in *European Conference on Machine Learning*. Springer, 2006, pp. 654–662.

[7] F. J. Gomez and R. Miikkulainen, "Solving non-markovian control tasks with neuroevolution," in *IJCAI*, vol. 99, 1999, pp. 1356–1361.

[8] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," in *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 1996, pp. 81–89.

[9] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[10] M. Lecesne, *Trophy, a 2D car racing action game for Linux.*, 2017. [Online]. Available: http://trophy.sourceforge.net/

[11] J. Mänttäri, "Applications of artificial neural networks in games; an overview."

[12] M. Parker and B. D. Bryant, "Neurovisual control in the quake ii environment," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 44–54, 2012.

[13] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *IEEE Transactions on Computational Intelligence and AI in Games*, 2015.

[14] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, "Neuroevolution for general video game playing," in *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*. IEEE, 2015, pp. 200–207.

[15] J. Schrum and R. Miikkulainen, "Constructing complex npc behavior via multi-objective neuroevolution." *AIIDE*, vol. 8, pp. 108–113, 2008.

[16] K. Stanley, "Neat c++ version 1.2.1," 2010. [Online]. Available: http://nn.cs.utexas.edu/?neat-c

[17] K. O. Stanley. (2017) Competitive coevolution through evolutionary complexification. [Online]. Available: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html

[18] K. O. Stanley and R. Miikkulainen. (2001) Efficient evolution of neural network topologies. [Online]. Available: http://nn.cs.utexas.edu/downloads/papers/stanley.cec02.pdf

[19] ——, "Evolving a roving eye for go," in *Genetic and Evolutionary Computation Conference*. Springer, 2004, pp. 1226–1238.

[20] M. E. Taylor, S. Whiteson, and P. Stone, "Transfer via inter-task mappings in policy search reinforcement learning," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, p. 37.