Neuroevolution of an Automobile Crash Warning System

Kenneth Stanley
Computer Sciences Dept
University of Texas, Austin
Austin, TX 78712-0233
kstanley@cs.utexas.edu

Nate Kohl
Computer Sciences Dept
University of Texas, Austin
Austin, TX 78712-0233
nate@cs.utexas.edu

Rini Sherony
Technical Research Dept
Toyota Technical Center
Ann Arbor, MI 48105
Rroy@ttc-usa.com

Risto Miikkulainen
Computer Sciences Dept
University of Texas, Austin
Austin, TX 78712-0233
risto@cs.utexas.edu

ABSTRACT

Many serious automobile accidents could be avoided if drivers were warned of impending crashes before they occurred. In this paper, a vehicle warning system is evolved to predict such crashes in the RARS driving simulator. The NeuroEvolution of Augmenting Topologies (NEAT) method is first used to evolve a neural network driver that can autonomously navigate a track without crashing. The network is subsequently impaired, resulting in a driver that occasionally makes mistakes and crashes. Using this impaired driver, a crash predictor is evolved that can predict how far in the future a crash is going to occur, information that can be used to generate an appropriate warning level. The main result is that NEAT can successfully evolve a warning system that takes into account the recent history of inputs and outputs, and therefore makes few errors. Experiments were also run to compare training offline from previously collected data with training online in the simulator. While both methods result in successful warning systems, offline training is both faster and more accurate. Thus, the results in this paper set the stage for developing crash predictors that are both accurate and able to adapt online, which may someday save lives in real vehicles.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Connectionism and neural nets*

General Terms

Experimentation

Keywords

neuroevolution, vehicle, warning, NEAT

1. INTRODUCTION

A significant goal of Artificial Intelligence research is to reduce human loss of life and injury. In principle, intelligent systems can warn humans and protect them from potentially dangerous situations. Evolutionary algorithms have the potential to identify danger where it might otherwise not be apparent by learning about dangerous situations through experience. In this paper, artificial neural networks are evolved to warn drivers in dangerous situations, showing that, at least in principle, learning may eventually help save lives.

If cars could warn their drivers that a crash is imminent, it is possible that many accidents could be avoided. One approach for building such a warning system is to ask an expert to describe as many dangerous situations as possible and formalize that information in an automated reasoner. However, the circumstances leading to a crash are frequently subtle and may vary for different drivers. Moreover, it may not be possible to predict a crash from a static snapshot of the road; the recent history of the car and other objects on the road may have to be taken into account as well. It is difficult to know how long such a history should be or what objects it should track.

Yet if the car could learn *on its own* what to track and how long to keep salient events in memory, these challenges could be overcome. In addition, cars could be trained with different drivers under different circumstances, creating more flexible warning systems.

Teaching a car to predict crashes is the goal of the automobile warning system project at the University of Texas at Austin, started in November 2003 and funded in part by Toyota. The NeuroEvolution of Augmenting Topologies (NEAT; [9, 10]) method was used to evolve crash prediction neural networks. NEAT is a natural choice for the learning method because it evolves the network topology in addition to the weights, and therefore can develop arbitrary recurrent neural networks that keep a variable length of prior history in memory. In other words, an expert does not need to decide how long the warning window should be or what it should take into account, because evolution makes this determination in selecting the appropriate recurrent topology. In addition, NEAT has shown promise in control tasks such as pole balancing and a simulated robot duel, suggesting it could make effective judgments about the danger of vehicle circumstances [9, 10]. Because NEAT matches the complexity of the neural network with the complexity of the task, it can find the right level of representation for warning under different conditions.

In order to evolve crash-predictor networks, a large set of examples of bad driving must be available. While it would be difficult to obtain such a set from human drivers, one convenient way is to evolve the drivers in simulation before evolving the warning networks. In fact, past successes with evolving controllers suggests that robust and varied driving behavior can be obtained in this manner, forming a solid foundation for evolving the warning networks. This is the approach taken in this paper.

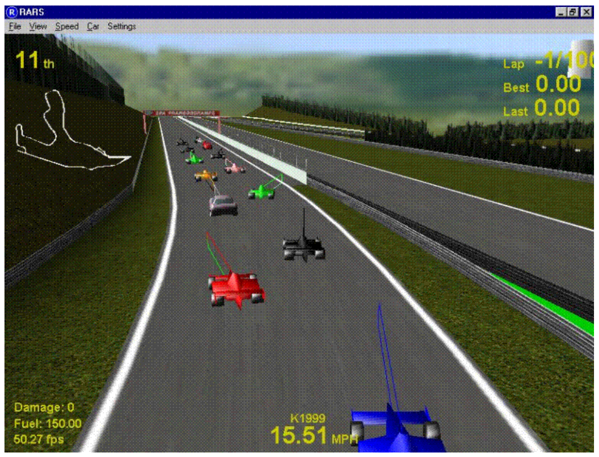
An important question is whether the warning system should be evolved online while the car is running, or offline with prerecorded data. This question is important because it may be necessary to train the system online in order to adapt to particular drivers and driving conditions. On the other hand, it would be easier and less expensive to train a warning network for the real world if driving data could be recorded from real drivers and later used for training. To determine whether these two training scenarios lead to different performance, warning networks were trained in both ways in this paper. In online training, the networks were evaluated by predicting crashes for cars as they drove inside the simulator. In offline training, a driver was first recorded for a certain amount of time in the simulator, and warning networks were later evaluated offline during evolution from the prerecorded training data. The results demonstrate that while both online and offline training can be used, learning from offline data is faster and produces more accurate warnings during testing. This suggests that an effective system can therefore be constructed through offline training and later adapted to particular conditions as necessary through online training.

The next section describes the RARS driving simulator used in the experiments. The NEAT neuroevolution method is described in section 3. Section 4 explains how automated drivers were trained, and Section 5 demonstrates how these drivers were used to evolve crash predictors. Finally, Section 6 describes the experimental results and Section 7 discusses the benefits of online and offline training and outlines future work.

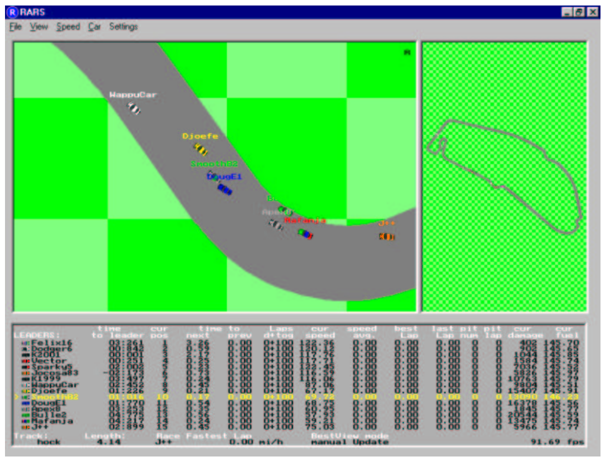
2. THE ROBOT AUTO RACING SIMULATOR (RARS)

Since learning requires experience, it is necessary for the learning system to gain experience through driving and predicting crashes. Because crashing cars in the real world would be dangerous and expensive, it is necessary to do it in simulation. RARS (`http://rars.sourceforge.net`; Figure 1), a public domain racing simulator designed for testing Artificial Intelligence (AI) methods for real-time control, is ideally suited for this purpose.

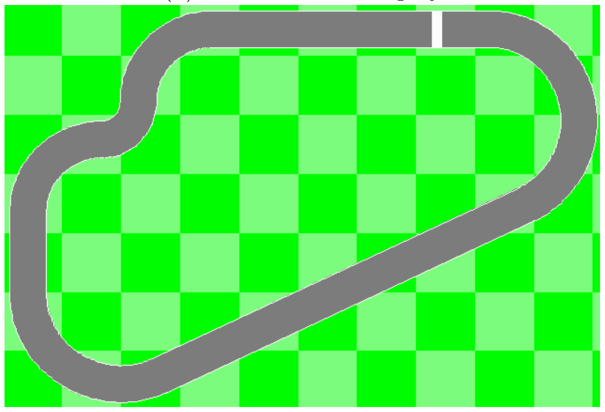
RARS is supported by an active community that provides documentation and software maintenance. The software was written with AI in mind, so it is easy to modify existing drivers and introduce new ones. Vehicle dynamics are accurately simulated, including skidding and traction. Multiple automobiles controlled by different automated drivers can



(a) 3-D Overhead View



(b) 2-D Overhead Display



(c) The “clkwis” track

Figure 1: Screenshots of the RARS driving simulator. The screenshots show three views of a race in RARS. (a) In the 3-D view, lines projecting in front of the cars show their current trajectories. (b) The 2-D view shows more of the track at the same time, and also a full map and current rankings. (c) A 2-D view of the “clkwis” track used for this research described in this paper. All of these views can represent the same race, which can include any number of cars operated by independent controllers. Because RARS is a popular platform that accurately simulates vehicle physics and supports multiple simultaneous drivers, it makes a good simulation testbed for evolving warning systems.

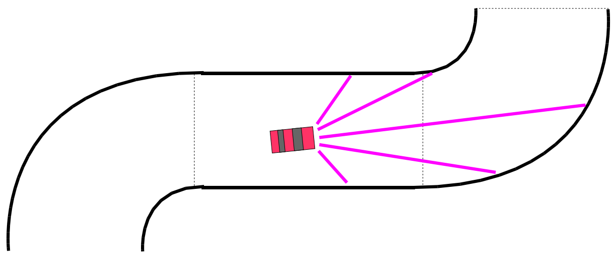


Figure 2: Rangefinder sensors in RARS. Simulated rangefinders project lines to the edges of the road and measure the distance to the turns, giving the car a sense of its position and the road's curvature. RARS' native data structures were converted into rangefinder data so that neural networks could be trained with a realistic egocentric input.

race at the same time. The software automatically provides information like the distance between the driver and other vehicles and the direction of the road that can be used as the basis for simulated sensors.

RARS driving data was converted into a rangefinder sensor representation that was provided to the neural network to sense road edges. Rangefinder sensors project rays at several angles relative to the car's heading to the edge of the road (Figure 2). The rangefinders give the car an indication of its position and heading relative to the sides of the road, and also of the curvature of the road.

It is sensible to ask whether this sensor configuration is a reasonable approximation of the real world, i.e. can similar information be extracted from real sensors? In fact, significant research has gone into detecting lanes in the real world [1]. Data from such systems could be processed and fed into the neural networks in a similar form to the sensors used in these experiments. Of course, in the future it will also be desirable to simulate more sophisticated processed sensors such as vehicle trackers [5, 11], but the current sensors form a reasonable, realistic starting point.

RARS provides a virtual gas pedal, brake, and steering wheel that can receive their values from the outputs of a neural network. The gas pedal and brake are interpreted as a requested tire speed relative to the bottom of the car. There is no limit to how high the request can be, and RARS tries to match the request within the physical constraints of the car. In addition, if the request is lower than the current speed, RARS attempts to slow the car down by braking. The steering request is treated similarly: the lateral force generated by the same turn angle request increases the higher the current speed. Thus, the driving controls in RARS work like a real automobile.

Races can be set up in RARS with one or more drivers. A natural way to begin training drivers and warning systems is on an open road without other cars. The experience and data gained from starting in this way can be used to support training more sophisticated warning systems with obstacles and other cars in the future.

The next section describes the evolutionary algorithm used to train both neural network drivers and warning systems.

3. NEUROEVOLUTION OF AUGMENTING TOPOLOGIES (NEAT)

It is not known how complex either a driving or warning network needs to be or even what kind of topology it should have. Searching in too large a space, i.e. a space of highly complex networks, would be intractable while searching in too simple a space would limit solution quality. Moreover, it is not known how many or where recurrent connections should exist in the network to allow it to react to past states. For example, if the car is skidding to one side, each snapshot of the sensory input looks perfectly normal. A skid can only be identified by combining the observations over the last several time steps.

The NeuroEvolution of Augmenting Topologies (NEAT) method [9], which automatically evolves network topology to fit the complexity of the problem, is designed to solve these problems. NEAT combines the usual search for the appropriate network weights with *complexification* of the network structure. It starts with simple networks and expands the search space only when beneficial, allowing it to find significantly more complex controllers than fixed-topology evolution. This approach is highly effective: NEAT outperforms other neuroevolution (NE) methods on complex control tasks like the double pole balancing task [9, 8] and the robotic strategy-learning domain [10]. These properties make NEAT an attractive method for evolving neural networks in complex tasks. In this section, the NEAT method is briefly reviewed; see [9, 8, 10] for more detailed descriptions.

NEAT is based on three key ideas. First, evolving network structure requires a flexible genetic encoding. Each genome in NEAT includes a list of *connection genes*, each of which refers to two *node genes* being connected. Each connection gene specifies the in-node, the out-node, the weight of the connection, whether or not the connection gene is expressed (an enable bit), and an *innovation number*, which allows finding corresponding genes during crossover. Mutation can change both connection weights and network structures. Connection weights are mutated in a manner similar to any NE system. Structural mutations, which allow complexity to increase, either add a new connection or a new node to the network. Through mutation, genomes of varying sizes are created, sometimes with completely different connections specified at the same positions.

Each unique gene in the population is assigned a unique innovation number, and the numbers are inherited during crossover. Innovation numbers allow NEAT to perform crossover without the need for expensive topological analysis. Genomes of different organizations and sizes stay compatible throughout evolution, and the problem of matching different topologies [7] is essentially avoided.

Second, NEAT speciates the population so that individuals compete primarily within their own niches instead of with the population at large. This way, topological innovations are protected and have time to optimize their structure before they have to compete with other niches in the population. The reproduction mechanism for NEAT is *explicit fitness sharing* [2], where organisms in the same species must share the fitness of their niche, preventing any one species from taking over the population.

Third, unlike other systems that evolve network topologies and weights [4, 12], NEAT begins with a uniform population of simple networks with no hidden nodes. New structure is

Driver	Lap Time
Apex8	**1:39**
SmoothB2	**1:33**
NEAT	**1:31**
Felix16	**1:22**
Bulle2	**1:18**

Table 1: Driving times for hand-coded and evolved drivers. Hand-coded drivers and a driver evolved by NEAT were timed on the "clkwis" track (Figure 1c) provided with RARS. The code for driver Apex8 was written by Maido Remm, Bulle2 by Marc Gueury, Felix16 by Doug Eleveld, and SmoothB2 by Dennis Lew. Each driver was timed on a single lap around the empty track. Even though the goal was not to evolve the fastest possible driver, NEAT's time is on par with the best hand-coded drivers.

introduced incrementally as structural mutations occur, and the only structures that survive are those that are found to be useful through fitness evaluations. In this manner, NEAT searches through a minimal number of weight dimensions and finds the appropriate complexity level for the problem.

In the automobile warning project, NEAT was used both to train drivers and crash predictors, as described in the next two sections.

4. TRAINING DRIVERS

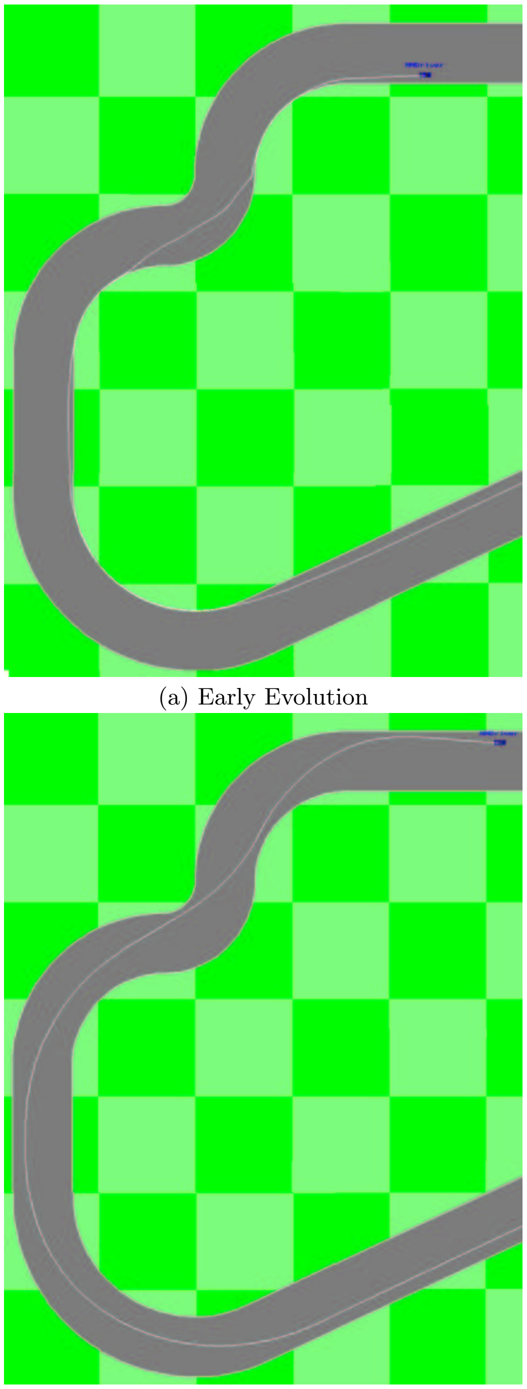
Training warning networks requires a driver from which to learn. It is important, however, that the driver used to train warning networks makes occasional mistakes and provides the warning networks with some real crash experience. Such a driver can easily be obtained by slightly perturbing the weights of a successful evolved neural network driver. This section explains how such drivers were evolved.

The drivers were evolved with seven rangefinder sensors. During evolution, each neural network in the population was evaluated over three trials. Each trial lasted 1,000 simulated timesteps, which is long enough to go around the track once. The network's fitness was the average score over the three trials. The score S for a single trial was

$$S = 2d - b, \tag{1}$$

where d is the distance traveled and b is the damage incurred during the trial. Damage is computed by RARS internally, and is proportional to time off the track. This fitness function penalizes crashing and rewards speed. The rest of the evolution parameters are described in section 6.1.

Using this methodology, skilled open road drivers were evolved that never crash. Although the purpose of this project is to evolve warning networks and not fast drivers, lap times for evolved drivers turned out to be comparable to the best hand-coded drivers provided with RARS (Table 1). Figure 3 shows the trajectory of a champion vehicle on the track at two different stages of evolution. The first successful networks to evolve learned to follow the inside of the track without going off the track. This method is effective but not particularly fast. Interestingly, later on drivers were evolved that utilized the entire width of the track, entering and exiting turns from the outside. Such a strategy allows drivers to maneuver the car through turns at higher speeds and leads


(a) Early Evolution

(b) Late Evolution

Figure 3: Discovering intelligent driving through a turn. (a) Early in evolution, the car hugs the inside of the turn, a naive strategy that minimizes driving distance but results in slow overall speed. (b) After approximately 400 generations, the driver learns to anticipate the turn by steering to the outside so it can attain maximum acceleration coming out of the turn. Notice the car takes an almost straight line through the most curvy section of the road. Although this behavior is counter-intuitive and hard even for humans to learn, it was discovered automatically through evolution.

to faster lap times. However, it increases the distance that the car must travel, and is therefore not an obvious strategy to learn. This surprising result shows that NEAT optimizes nontrivial behavior and discovers sophisticated techniques on its own.

After a skilled driver was evolved, it was intentionally impaired in order to obtain a driver that behaved mostly rationally but crashed intermittently. The weights of the champion driver were perturbed using uniform random noise between -0.4 and 0.4. The resulting driver could still navigate the track, but occasionally made erroneous decisions and ended up crashing. This driver was then be used to establish a good training environment for learning to warn about impending crashes.

5. TRAINING CRASH PREDICTORS

The crash predictor neural network receives the same inputs as the driving network. However, instead of outputting driving control requests, its task is to *predict* whether and when a crash is going to happen. This prediction is based on the sensor inputs over several time steps, describing the dynamics of the situation. If the predictor has a good model of the driver's behavior, it can make realistic predictions about what the driver is likely to do in the current situation, and therefore predict whether a crash is likely to happen.

Importantly, the networks evolve to determine on their own how many timesteps in the past are necessary to observe. The recurrent structures are selected during evolution based on how well they support the predictions.

The simplest kind of prediction is a binary output that predicts whether or not a crash will happen in some fixed number of timesteps. While such a system is useful, a more sophisticated prediction can be made if the network also determines *when* it expects the crash. By predicting a time, the network is in effect constantly outputting a danger level: the sooner the predicted crash, the more dangerous the situation. Such a graded warning system is likely to be more useful to human drivers, allowing e.g. different warning signals to be used depending on their urgency.

The temporal prediction network is given the same rangefinder inputs as the driving networks (Figure 2). The network has a single output that is interpreted as a predicted time to crash between zero (i.e. imminent crash) and a maximum time m. In general, when the network outputs the maximum value, it means there is no present danger. Fitness is computed as the mean squared error $\overline{E}$, accumulated while the impaired driver drives around the track. Let I_t be the correct prediction at timestep t and o_t be the prediction output by the network. In the event a crash is more than m timesteps in the future, I_t is set to m. In computing $\overline{E}$, I_t and o_t, which range between zero and m, are scaled between zero and one. The mean squared error $\overline{E}$ over n timesteps is then:

$$\overline{E} = \frac{\sum_{t=1}^{n} (o_t - I_t)^2}{n}. \tag{2}$$

Warning networks were evaluated both offline from prerecorded training data and online using the RARS simulator for each evaluation. Offline data was generated from the same impaired driver that drove in online evaluation. In the offline case, $\overline{E}$ is computed by comparing each prediction to a precomputed set of ideal prediction targets I_1 through I_n. In the online case, predictions are stored in a queue while the driver runs in the simulator. At every timestep, the network output is pushed onto this prediction queue, which becomes a moving window of past predictions. Ideally, if a crash happens, the prediction queue has low-level warnings farther back in time and high-level warnings more recently (Figure 4). Each time a crash occurs, the correct predictions are computed for each of the previous m timesteps and compared with the stored predictions. This process is repeated until the end of the evaluation. In this way, both offline and online training can use the same fitness criterion.

Using temporal predictions, it is possible to evolve networks that vary their warning level. The next section specifies the major research questions, describes the experiments that were set up to answer them, and presents the results.

6. EXPERIMENTS

The primary aim of this research is to show that networks can be evolved to reliably warn about imminent crashes. Both online and offline training could be useful in constructing such a warning system. Offline training using a prerecorded dataset could be used to ensure that the system achieves an acceptable performance in most common situations; online training could allow the system to adapt to new drivers and conditions. The problem with online training is that since it is so noisy, a standalone online learning system may not be given consistent enough feedback to be able to successfully train precise crash predictors. On the other hand, there may not be enough variation in offline training to allow robust and accurate warning networks to develop.

Showing that a warning system can be learned in either case and answering some preliminary questions about how the two approaches differ sets the groundwork for future research in automated warning systems. The following sections describe the experiments used to evaluate both online and offline training and presents results showing how the two approaches fared.

6.1 Experimental Setup

During offline runs, sensor values, outputs, and crash times were recorded from the impaired champion driver. A total of 8541 timesteps and 150 crashes were collected. The data was then processed so that a target time-to-crash, I_t, was associated with each timestep. In other words, if a crash occurred within the maximum time $m = 75$, I_t was the number of timesteps until the crash. Otherwise, I_t was set to 75 (i.e. about the time it takes to traverse 5% of the track), denoting that the crash is far in the future. During training, the output of the neural network was compared to I_t (scaled between zero and one) at each timestep in order to compute an error. As described in Section 5, the mean squared error over all the training data was then taken as the network's fitness.

The entire 8541 timestep dataset was randomly split into three subsets: one for training (70% of the dataset), one for validation (20%), and one for testing (10%). Each subset was composed of contiguous data because the data points represent a chronology of events. In order to perform 10-fold cross-validation, 10 such three-way splits were made, each chosen randomly. The validation set was used to determine which generation champion to use during testing. If there was no improvement in performance on the validation

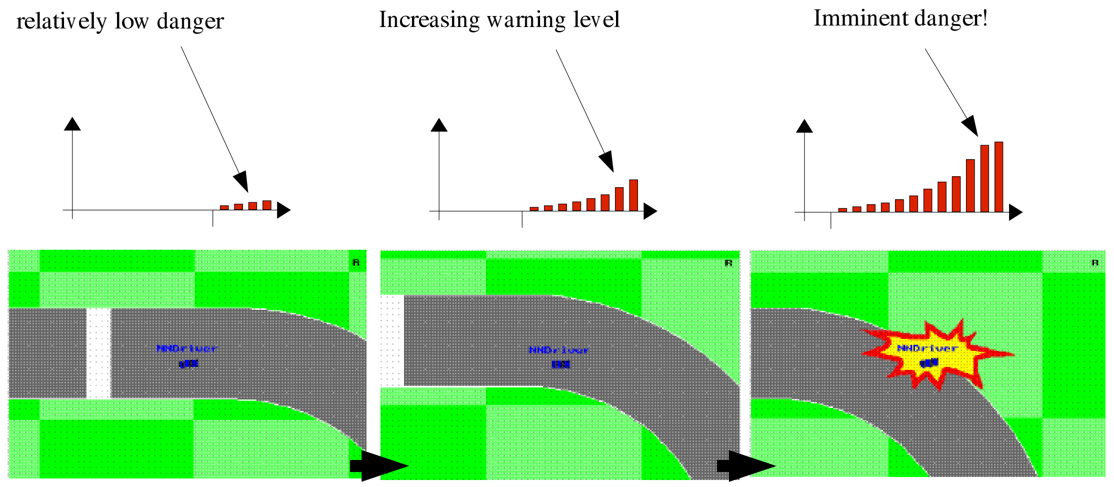


Figure 4: Representing an impending crash through the prediction queue. The most recent warning appears on the right of the queue and the higher bars represent increasing urgency. In other words, the less time to a predicted crash, the higher the bar is. As the car moves closer to crashing into the side of the road, the warning bars increase in size. The prediction queue allows training neural networks to predict not only that a crash is going to occur, but the time to crash as well.

set in 50 generations, the champion from the first of those generations was utilized for testing.

In contrast to the offline runs, in the online runs each neural network attempted to make predictions during the live RARS simulation. The same impaired driver was used in online training as in offline training, and the mean squared error was computed just as in the offline runs by keeping data in a queue (Section 5). Since RARS is not deterministic (because of noise added to the actuators), the same controller on the same track will not always perform exactly the same. Commands to move the steering wheel or increase the gas can have slightly variable consequences, just as in the real world, and the crashes are likely to occur in different places. Thus, the main difference between training offline and online is that online evaluations varied even with the same driver. After training, the online champion networks were tested with the same testing sets as the offline networks.

Because population dynamics can be unpredictable over hundreds of generations, a target of 5 species in the population of 100 networks was assigned to NEAT evolution. The champion of each species with more than five networks was copied into the next generation unchanged. The interspecies mating rate was 0.05, the probability of adding a new node was 0.03, and the probability of a new link mutation was 0.05. These parameters were found through systematic experimental search.

6.2 Results

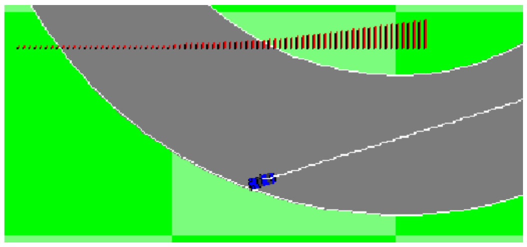
Successful predictors evolved in all simulation runs. In some situations, the warning network predicted crashes that could not be predicted only from the current state of the car. For example, when the car skids into the side of the road, its heading is in a direction that could be interpreted as safe. Yet the evolved network still predicts a crash, showing that it is using memory to integrate a sequence of states into its prediction. The warning system can also be evaluated subjectively by having a human drive the car with the warning system on. Human drivers generally found the warnings accurate and helpful. Figure 5 shows actual prediction queues generated in real-time during human-controlled driving tests.

As expected, training offline was significantly faster than online training. On a Pentium 4 1.8GHz processor, offline training took on average about 400 minutes to compete 500 generations, compared to $1,400$ minutes for online training. The overhead of running the simulator online accounts for the difference.

In order to make sure that the performance levels achieved by both offline and online learning were not the result of trivial strategies such as never warning or warning all the time, those trivial strategies were also evaluated on the test dataset. In addition, a predictor that guesses a uniform random time to crash between 0 and 75 was included in the comparisons.

Figure 6 summarizes the results. First, both online and offline training performed significantly better than the three trivial baseline strategies ($p =< 0.001$). The main result is, therefore, that learning indeed produces meaningful warning behavior in both cases. Second, the mean squared error for predictors trained offline was on average 0.054 less than for those trained online training. This difference is significant ($p < 0.001$), suggesting that the natural variation in online training is slightly detrimental in this task.

An analysis of error over time confirms this conclusion. The average generation in which performance on the validation set stops improving is 93.6 ($sd = 31.0$) in online training and 342.3 ($sd = 118.5$) in offline training. In other words, offline learning continues an average of 248.7 generations longer. This difference is significant with $p < 0.001$. After the most basic warnings have been learned, the noise in the online training prevents minor improvements from being evaluated accurately, and evolutionary progress stops. In contrast, offline training provides consistent evaluation from generation to generation, thereby allowing even minor


(a) Direct crash

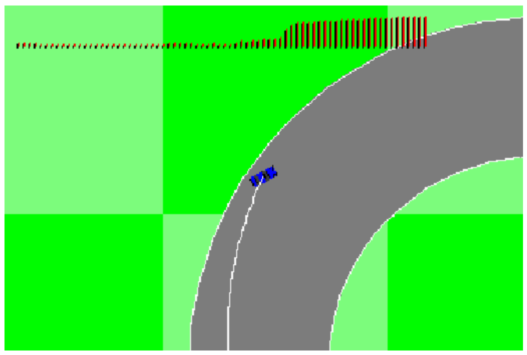

(b) Skidding

Figure 5: Warning examples. The real-time prediction queues are shown at the top of each image with the most recent prediction on the right. The height of the bars represents the seriousness of the warning. (a) As the car drives directly into the side of a turn, the warning switches from mild to severe. (b) Judging by the car's heading alone, there would be no reason to predict a crash in this scenario. However, as can be seen by the white trajectory line preceding the car, it has been skidding sideways for some time. The predictor network was able to make the right warning by integrating the information over several timesteps.

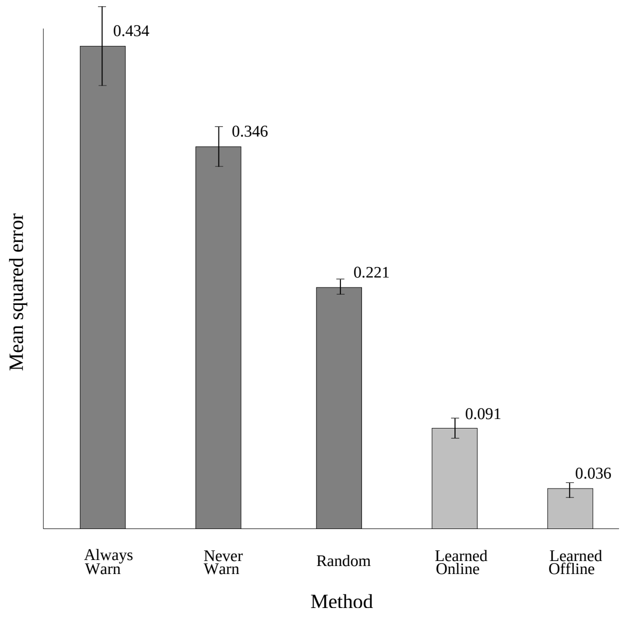


Figure 6: Performance of online and offline training methods compared to baseline strategies. Ten runs using each method were performed; the graph shows the mean squared error, $\overline{E}$, and one standard deviation for all methods. In *online* training, all evaluations during training took place inside the slightly non-deterministic simulator. In *offline* training the examples were collected earlier and were fixed throughout evolution. The performance of three trivial baseline strategies are also reported: *Random* made a uniform random guess for the time to crash at every timestep, *never warn* always predicted that no crash was imminent, and *always warn* consistently warned that a crash was about to occur. The differences between offline and online methods, and between both offline and online and the baseline methods, are statistically significant ($p < 0.001$). The main results are that evolution was able to generate meaningful warnings and that offline evolution performed better than online evolution.

improvements to enter the population. This result is interesting because noisy simulations have previously been shown to lead to more robust behaviors [3, 6]. The conclusion is that in order to obtain robust and accurate warnings, variation in training scenarios has to be carefully controlled, as proposed in the next section.

7. DISCUSSION AND FUTURE WORK

Evolving automobile warning systems is a promising goal for evolutionary computation because the task is based on subtle but learnable correlations, and any improvement over random guessing can potentially save lives and property. NEAT is a natural choice for generating such systems because it can evolve both drivers and predictors, and it can automatically determine what information is useful in making the decisions. Results in this paper also uncovered an important difference between learning while driving is happening and learning from data collected offline.

Since offline training proved both faster and produced higher quality results, the question arises whether training in a simulator, or even in a real car, is ever going to be useful. The answer is that it serves a different purpose. While offline training can be used for generating a robust and general warning system, it is not possible to anticipate all possible situations while collecting data. The system still needs to be augmented and refined, depending on the needs of each particular case. The cars, conditions, and driver ability vary, but each individual case provides an evaluation that is relatively consistent. It should therefore be possible to customize the warning system through online learning.

Two types of errors contribute to the total error of the crash predictor: Either a crash is predicted that never occurs, or a crash occurs that was not predicted. An interesting opportunity for future research is to weigh the two types of errors differently so that the system is biased towards making fewer of the more expensive kind of mistake at the cost of making more of the less expensive type. For example, hearing a beep in an otherwise safe situation is little more than irritating, but not being warned before a serious crash can be fatal. Because evolutionary algorithms can weight different error components differently in the fitness function, they can potentially calibrate the system to have a bias towards being safe over sorry.

In the future, the warning system will be trained in more complex environments including obstacles and other cars on the road. As of the writing of this paper, a driver has already been evolved that can weave around stationary cars parked on a track. Once impaired, this driver will be used to train the crash predictor about dangerous situations with other cars. Ultimately, as the technology progresses, systems like the crash predictor in this paper may be deployed in the real world.

8. CONCLUSION

The NEAT neuroevolution method was used to evolve both drivers and crash predictors on the open road in the RARS driving simulator. Crash predictors output an estimated time to crash that can be used to gauge the current danger level. The best predictors were evolved from driving data collected offline, exhibiting a strong performance of on average under 0.04 mean squared error in predicted time to crash. Successful crash predictors were also evolved online in simulation suggesting that online evolution could be used to customize the warning system to particular cars, drivers, or conditions. The main conclusion is that automatic crash prediction is possible, and may someday save lives in the real world.

9. ACKNOWLEDGMENTS

The authors would like to thank all of the programmers that contributed to the open-source RARS project, which is available at `http://rars.sourceforge.net`. This research was supported in part by a grant from Toyota, USA.

10. REFERENCES

[1] E. Dickmanns and B. Mysliwetz. Recursive 3-D road and relative ego-state recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):199–213, 1992.

[2] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154. San Francisco: Kaufmann, 1987.

[3] F. J. Gomez and R. Miikkulainen. Transfer of neuroevolved controllers in unstable domains. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2004)*, Berlin, 2004. Springer Verlag.

[4] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 81–89, Cambridge, MA, 1996. MIT Press.

[5] M. Haag and H.-H. Nagel. Combination of edge element and optical flow estimates for 3d-model-based vehicle tracking in traffic image sequences. *International Journal of Computer Vision*, 35(3):295–319, 1999.

[6] S. Nolfi and D. Floreano. *Evolutionary Robotics*. MIT Press, Cambridge, 2000.

[7] N. J. Radcliffe. Genetic set recombination and its application to neural network topology optimization. *Neural computing and applications*, 1(1):67–90, 1993.

[8] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, San Francisco, 2002. Kaufmann.

[9] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[10] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004.

[11] F. Thomanek, E. D. Dickmanns, and D. Dickmanns. Multiple object recognition and scene interpretation for autonomous road vehicle guidance. In *Intelligent Vehicles Symposium '94*, 1994.

[12] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.