Improving Artificial Intelligence In a Motocross Game

Benoit Chaperot
School of Computing
University of Paisley
Scotland
benoit.chaperot@paisley.ac.uk

Colin Fyfe
School of Computing
University of Paisley
Scotland
colin.fyfe@paisley.ac.uk

Abstract— **We have previously investigated the use of artificial neural networks to ride simulated motorbikes in a new computer game. These artificial neural networks were trained using two different training techniques, the Evolutionary Algorithms and the Backpropagation Algorithm. In this paper, we detail some of the investigations to improve the training, with a view to having the computer controlled bikes performing as well or better than a human player at playing the game. Techniques investigated here to improve backpropagation are bagging and boosting, while alternative crossover techniques have also been investigated to improve Evolution.**

Keywords: Motorbikes, Computational Intelligence, Artificial Neural Networks, Back Propagation, Evolutionary Algorithm, Genetic Algorithm, Driving Game.

I. Introduction

We [1] have previously investigated the use of artificial neural networks to ride simulated motorbikes in a new computer game. In this paper, we investigate techniques for improving the training of artificial neural networks to ride simulated motorbikes in a new computer game. The use of such methods in control is not new (see e.g.[2], [3]), but it is one of the first time these methods are applied to a video game (see e.g.[4]). Two training techniques are used, Evolutionary Algorithms and the Backpropagation Algorithm. To improve the Backpropagation Algorithm in this paper, two optimisation techniques for augmenting the training are used: bagging [5] and boosting [6]. There are various interesting aspects in using artificial neural networks (ANN's) in a motocross game. The main reason is that, although the control of the bike is assisted by the game engine, turning the bike, accelerating, braking and jumping on the bumps involve behaviours which are difficult to express as a set of procedural rules, and make the use of ANN's very appropriate. Our main aim is then to have the ANN's to perform as well as possible at riding the motorbike. We suspect that we can train the network to play better than any living player; however, an ANN can always be penalised at a later stage if it becomes so good that it decreases the enjoyment of human competitors.

II. The Game

There are various interesting aspects in using artificial neural network methods in a motocross game. Because the design of an ANN's is motivated by analogy with the brain, and the rationale for their use in the current context is that entities controlled by ANN's are expected to behave in a human or animal manner, these behaviours can add some life and content to the game. The human player has also the possibility to create new tracks. ANN's have the capability to perform well and extrapolate when presented with new and different sets of inputs from the sets that were used to train them; hence an ANN trained to ride a motorbike on a track should be able to ride the same motorbike on another similar track. ANN's are adaptible in that their parameters can be trained or evolved. ANN's may be able to perform with good lap times on any given track while still retaining elements of human behaviour.

Motocross The Force is a motocross game featuring terrain rendering and rigid body simulation applied to bikes and characters. An example of it in use can be seen at

`http://cis.paisley.ac.uk/chap-ci1`

and a screen shot from the game is shown in Figure 1. The game has been developed and is still being developed in conjunction with Eric Breistroffer (2D and 3D artist). A track has been created in a virtual environment and the game involves riding a motorbike as quickly as possible round the track while competing with other riders who are software-controlled.

There is one position known as a way point which marks the position and orientation of the centre of the track, every metre along the track. These way points are used to ensure bikes follow the track and we will discuss positions in way point space when giving positions with respect to the way points.

For example, for the evolutionary algorithms, the score is calculated as follows:

- **vPassWayPointBonus** is a bonus for passing through a way point.
- **vMissedWayPointBonus** is a bonus/penalty (i.e. normally negative) for missing a way point.
- **vCrashBonus** is a bonus/penalty (i.e. normally negative) for crashing the bike.
- **vFinalDistFromWayPointBonusMultiplier** is a bonus/penalty (i.e. normally negative) for every metre away from the centre of the next way point.

The inputs to the ANN are:

- Position of the bike in way point space.

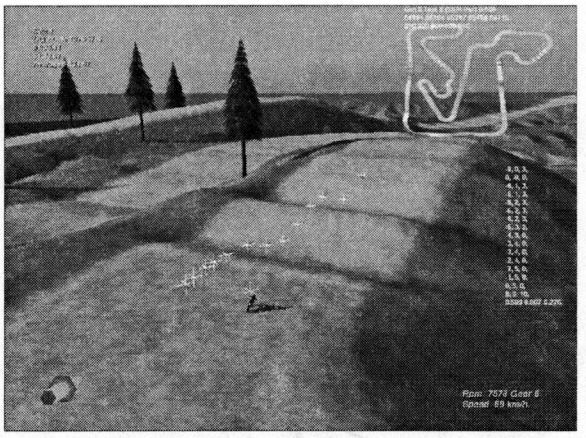

Fig. 1. Screen shot taken from the game; the white crosses represent the position of the track centre lane; there are 13 samples which are used as inputs to the ANN.

- Front and right directions of the bike in way point space.
- Velocity of the bike in way point space.
- Height of the ground, for $\mathbf{b}$ (typically 1) ground samples, in front of the bike, relative to bike height.
- Position of track centre lane, for $\mathbf{c}$ (typically 13) track centre lane samples, in front of the bike, in bike space.

The outputs of the ANN are the same as the controls for a human player:

- Accelerate, decelerate.
- Turn left, right.
- Lean forward, backward

III. Artificial Neural Networks

Artificial neural networks are usually software simulations which are models at some level of real brains. We will, in this paper, use multilayered perceptrons (MLP) though other types of neural networks [7] may be equally useful for the task in this paper.

The MLP consists of an input layer, $\mathbf{x}$, whose neurons are passive in that they merely hold the activation corresponding to the information to which the network must respond. In our case this will be local information about the terrain which the artificial rider is currently meeting. There is also an output layer, $\mathbf{y}$, which in our case will correspond to the actions (turn left/right, accelerate/decelerate, lean forward/backward) which are required to ride the bike. Between these two layers is the hidden layer of neurons which is so-called as it cannot directly communicate in any way with the external environment; it may only be reached via the input neurons and only affects the environment via the output neurons.

The MLP is used in two phases: activation passing and learning. Activation is passed from inputs to hidden neurons through a set of weights, W. At the hidden neurons, a nonlinear activation function is calculated; this is typically a sigmoid function, e.g. $\frac{1}{1+\exp(-act)}$ which mimics the saturation effects on real neurons. Let us have N input neurons, H hidden neurons, and O output neurons. Then the calculation at the hidden neurons is:

$$act_i = \sum_{j=1}^{N} W_{ij}x_j, \forall i \in 1, ..., H$$

$$h_i = \frac{1}{1+\exp(-act_i)}$$

where h_i is the firing of the i^{th} hidden neuron. This is then transmitted to the output neurons through a second set of weights, V, so that:

$$act_i = \sum_{j=1}^{H} V_{ij}h_j, \forall i \in 1, ..., O$$

$$o_i = \frac{1}{1+\exp(-act_i)}$$

Thus activation is passed from inputs to outputs. The whole machine tries to learn an appropriate mapping so that some function is being optimally performed. Such networks use supervised learning to change the parameters, W and V i.e. we must have a set of training data which has the correct answers associated with a set of input data. The most common method is the backpropagation algorithm.

In the experiments discussed in this paper, we used the same activation function at the outputs as at the hidden neurons.

IV. The Backpropagation Algorithm

Let the P^{th} input pattern be $\mathbf{x}^P$, which after passing through the network evokes a response $\mathbf{o}^P$ at the output neurons. Let the target value associated with input pattern $\mathbf{x}^P$ be $\mathbf{t}^P$. Then the error at the i^{th} output is $E_i^P = t_i^P - o_i^P$ which is then propagated backwards (hence the name) to determine what proportion of this error is associated with each hidden neuron. The algorithm is:

1) Initialise the weights to small random numbers
2) Choose an input pattern, $\mathbf{x}^P$, and apply it to the input layer
3) Propagate the activation forward through the weights till the activation reaches the output neurons
4) Calculate the δs for the output layer $\delta_i^P = (t_i^P - o_i^P)f'(Act_i^P)$ using the desired target values for the selected input pattern.
5) Calculate the δs for the hidden layer using $\delta_i^P = \sum_{j=1}^{N} \delta_j^P w_{ji}.f'(Act_i^P)$
6) Update all weights according to $\Delta_P w_{ij} = \gamma.\delta_i^P.o_j^P$
7) Repeat steps 2 to 6 for all patterns.

An alternative technique for computing the error in the output layer while performing backpropagation has been investigated. Instead of computing the error as $(t_i^P - o_i^P)$, the error has been computed as $(t_i^P - o_i^P)|t_i^P - o_i^P|$. This has for effect to train the ANN more when the error is large, and allow the ANN to make more decisive decisions, with regard to turning left or right, accelerating or braking and leaning forward/back.

The backpropagation algorithm in the context of this motocross game requires the creation of training data made

from a recording of the game played by a good human player. The targets are the data from the human player i.e. how much acceleration/deceleration, left/right turning and front/back leaning was done by the human player at that point in the track. The aim is to have the ANN reproduce what a good human player is doing. The human player's responses need not be the optimal solution but a good enough solution and, of course, the ANN will learn any errors which the human makes.

We investigated two techniques to improve the training, bagging and boosting.

V. Ensemble methods

Recently a number of ways of combining predictors have been developed e.g. [8], [9], [6], [10]. Perhaps the simplest is bagging predictors. The term "bagging" was coined by joining bootstrapping and aggregating- we are going to aggregate predictors and in doing so we are bootstrapping a system. We note that the term "bootstrapping" was derived from the somewhat magical possibilities of "pulling oneself up by one's bootstraps" and the process of aggregating predictors in this way does give a rather magical result - the aggregated predictor is much more powerful than any individual predictor *trained on the same data*. It is no wonder that statisticians have become very convincing advocates of these methods.

A. Using Bagging

In ([1]), the ANN was trained using training data made from a recording of the game being played by a good human player. The data was made from the recording of the first author playing the game on many different motocross tracks (here 10). Bootstrapping [8] is a simple and effective way of estimating a statistic of a data set. Let us suppose we have a data set, $D = \{\mathbf{x}_i, i = 1, \ldots, N\}$. The method consists of creating a number of pseudo data sets, D_i, by sampling from D with uniform probability with replacement of each sample. Thus each data point has a probability of $(\frac{N-1}{N})^N \approx 0.368$ of not appearing in each bootstrap sample, D_i. Each predictor is then trained separately on its respective data set and the bootstrap estimate is some aggregation (almost always a simple averaging) of the estimate of the statistic from the individual predictors. Because the predictors are trained on slightly different data sets, they will disagree in some places and this disagreement can be shown to be beneficial in smoothing the combined predictor. Typically, the algorithm can be explained as follows:

1) Create N bags by randomly sampling from the data set with replacement.
2) The probability for a piece of data to be in the bag is approximately 0.63.
3) ANN's are trained on the bags separately.
4) The trained ANN's are then presented with an input and the outputs of the ANN's are combined.

TABLE I
Bagging Results

NN	Trained On Track	Training Data Length (sec)	Lap Time Track m16
0	Expert	234.41	1'58"
1	hat	144.42	6'43"
2	hillclimb	35.36	4'01"
3	jat	339.69	4'24"
4	L	261.51	2'22"
5	m1	247.32	4'45"
6	m10	131.72	NA
7	m11	187.47	14'25"
8	m12	100.62	4'08"
9	m13	112.06	6'21"
Average (0,9)	ALL	ALL	1'38"
11	ALL	1794.58	1'31"
Good Human	ALL	NA	1'23"

The combination operator, in spirit similar to [10], used was as below:

$$O_{ANN} = O_{AVE} * (1 - w) + O_{WIN} * w; \qquad (1)$$

With O_{AVE}, the average of all ANN's outputs, O_{WIN}, the output of the most confident ANN, which is the output with the largest magnitude, and w a parameter varying from 0 to 1. Experiments were done using ten ANN's. Experiments have shown that:

1) With $w = 0$, the combined output was a smooth output, and the computer controlled bikes tended to ride in a slow but safe manner.
2) With $w = 1$ (similar to "bumping" [10]) , the combined output was a decisive output and the computer controlled bikes tended to ride in a fast but risky manner.

This w parameter can allow to change the computer controlled bike behaviour, and could be used to have the artificial intelligence performance match that of the player. However, experiments proved that whatever the value for w, the performance was still a lot less than that of a good human player, and similar to or less than that of a single ANN trained using the entire training set.

Better results were achieve by, instead of creating bags by randomly sampling from data set, creating bags by sampling data according to data origin i.e. from data from a single track. The data set was made from the recording of the first author playing the game on ten motocross tracks. Now each bag contained data for one separate motocross track. The results are given in Table I.

(1) was used for combination with w equal to zero (pure averaging which is exactly bagging) and Table I shows that the combination of ten ANN's is better than every single ANN taken separately, but still not as good as another ANN trained using the full data set.

B. Using Boosting

There has been recent work identifying the most important data samples [11]; and presenting the ANN more with the

most important data samples (boosting [6]). We investigated the effect of different types of training data. For example, some parts of the track are relatively easy and the rider can accelerate quickly over these while other parts are far more difficult and so more care must be taken. The latter parts are also those where most accidents happen. Our first conjecture was that training the neural network on these more difficult parts might enable it to concentrate its efforts on the difficult sections of the track and so a training routine was developed in which each training sample has a probability to be selected for training the ANN proportional to the error produced the last time the sample was presented to the ANN. This allows us to train the ANN with more difficult situations.

The first algorithm was not efficient; it was storing an average error value for each of the training samples, which is not memory efficient, and made use of a roulette to select the samples according to the average error, which is not processing efficient. There was a time during which the average error was computed, and then the average error was reset. The algorithm was complicated, was making use of many parameters and was hard to tune.

Then, it appeared that because the learning rate is low, the ANN does not change very much with time, and it is possible to use the instantaneous error, and not the average error. Instead of selecting samples according to the error the sample produces, it is possible to select samples randomly, evaluate the error, and modify the instantaneous learning rate according to this instantaneous error.

The training routine had a negative effect on the training. Without the routine, the average lap time was 2 minutes and 40 seconds. With the routine, the average lap time was 3 minutes. On the other hand, when the routine was inverted (so that the backpropagation was performed with a learning rate proportional to the inverse of the error produced by the sample) this had a positive effect. The ANN performed better when being trained more with the easy samples.

Finally, the learning rate multiplier (to compute the effective learning rate from the original learning rate) was evaluated as:

$$m = MIN(0.1, 1 - Error); \quad (2)$$

Using this technique, after 24 hours of training, or 34560000 iterations, the ANN average lap time can go down from 2 minutes 30 seconds on track L, to only 2 minutes 18 seconds. Training can take a long time, because the ANN has to train and select the right set of training samples at the same time.

Our alternative technique for computing the error in the output layer while performing backpropagation as $(t_i^P - o_i^P)|t_i^P - o_i^P|$ was originally considered as having a positive effect on the training because it allowed more decisive decisions from the ANN, and proved to improve trained ANN performance. Since the time this alternative technique was implemented, the first author worked on the physics side of the game and the handling of the motorbike has improved; this alternative technique does not any more have a positive effect on the final performance of the ANN. Worse, it can have a negative effect.

It finally appeared that with the new physics, reverting to the classic technique for computing the error in the output layer, and removing the anti-boosting as described above allowed the ANN to train faster and produced equally good performances.

VI. Evolutionary Algorithms

We can identify the problem of finding appropriate weights for the MLP as an optimisation problem and have this problem solved using the GA ([12]): we must code the weights as floating point numbers and use the algorithm on them with a score function.

The algorithm is:

1) Initialise a population of chromosomes randomly.
2) Evaluate the fitness of each chromosome (string) in the population.
3) For each new child chromosome:
   a) Select two members from the current population. The chance of being selected is proportional to the chromosomes' fitness.
   b) With probability, C_r, the crossover rate, cross over the numbers from each chosen parent chromosome at a randomly chosen point to create the child chromosomes.
   c) With probability, M_r, the mutation rate, modify the chosen child chromosomes' numbers by a perturbation amount.
   d) Insert the new child chromosome into the new population.
4) Repeat steps 2-3 till convergence of the population.

An alternative technique for crossover has also been investigated: instead of crossing over the numbers (corresponding to the ANN's weights) from each chosen parent chromosome at a randomly chosen point to create the child chromosomes, numbers from parents are averaged to create the child chromosomes. This seems appropriate because we are working with floating point numbers and not binary digits and is a method which is sometimes used with the Evolution Strategies [13] which are designed for use with floating point numbers. Initial experimentation revealed that a blend of these two techniques worked best. The particular crossover technique was chosen randomly, with each technique being given equal chance, for each new child chromosome and then applied as usual.

Other techniques have also been tested: if starting the evolution from a randomly initialised chromosome population, we showed ([1]) that the ANN's do not perform as well as other ANN's trained using backpropagation. Some tests have been made starting the evolution from a population of different ANN's already trained using BP. One major problem with doing crossover with ANN's is that each neuron has a functionality or part of the behaviour (for example turning right), and while doing crossover, the child

chromosome may end up having twice the required number of neurons for a given functionality (turning right) and no neurons for another functionality (turning left). An attempt has been made, to reorder neurons in the parent ANN's, according to similarities and apparent functionalities, just before performing crossover, in order to reduce the problem. This proved not to be successful, and ANN's generated by the crossover of two very different ANN's still produced bad random behaviours. Eventually, because of elitism, and because crossover is not always performed, the population converges towards one individual ANN, which is not always the best one, and diversity in the population is lost. The best way to solve the problem was to start with one individual already trained ANN's, and mutate it to generate a starting population of differently mutated individuals. This proved very successful.

Our ANN's have 50 inputs, one hidden layer, 80 neurons in this hidden layer and 3 outputs. The number of weights to optimise is therefore 80*50+3*80=4240. Evolution can take a very long time to optimise all those weights. One optimisation technique was to discard in a early stage individuals which evaluate to be unfit, for example if the bike is in an unrecoverable situation. This considerably reduces the training time. However this optimisation can also sometimes evaluate fit individual as not being fit. This optimisation has therefore been removed and all ANN's have been given the same fixed evaluation time.

The number of cuts for crossover has been increased from one to ten; this means up to 11 different parts of the chromosomes can be swapped between parents to create the child chromosomes. This allows after only one generation combinations of chromosomes that would not have been possible with only one cut. The cuts are also made on the neurons' boundaries.

Six bikes are racing along track L, and therefore six ANN's are evaluated at any given time. The evaluation time has been set to 10 minutes, which means 30 minutes per generation. Currently computer controlled bikes don't see each other, and collision between bikes has been disabled in order not to have bikes interfere with one another.

The number of generations has been set to 100, with a population of 18 ANN's, elitism of 0 (number of the fittest chromosomes being passed directly from the parent population to the child population), a mutation rate of 0.001, a crossover rate of 0.8, a perturbation rate of 0.5, probability to select average crossover over 10 cuts crossover set to 0.2.

The training can take a long time to perform; however there are big advantages in the evolutionary algorithm approach. The artificial intelligence can adapt to new tracks and improve lap times with time; it is also possible that it can eventually perform better than a good human player.

Using this technique, after 24 hours of training, ANN's average lap time can go down from 2 minutes 45 seconds on the long track, to approximately 2 minutes 16 seconds. Not all individuals in the population are performing equally well. For comparison a good human player lap time is 2 minutes 10 seconds.

VII. More Improvements

The bike is moving and rotating a lot along the track. It appeared that instead of expressing the position of track centre lane in bike space, it was better to express it in forward space; with forward being the direction of the velocity vector. There are two main advantages in using the forward space instead of the bike space to transform ground samples:

1) It does not rotate in time in relation to the ground as much as the bike transform, so it allows the ANN to more easily identify input patterns for ground samples.
2) Because the velocity direction is now contained in the forward space used to transform ground samples, it is now possible to express the velocity as a scalar and not a vector and save two inputs for the ANN.

The bike maximum velocity was set slightly less for computer bikes than for the human player bike (30m/s against 32m/s). This reduction in maximum velocity proved to have a positive effect on the performance of computer bikes, because it prevented many accidents, at a time where ANN's were not performing well. Now the ANN's are performing better; this reduction is considered to have a negative effect and is removed.

Some new training data is created by having the first author playing the game on the long track for 23 minutes (138000 samples), with average lap times of approximately 2 minutes 8 seconds. The first author could have tried to optimise the training set, for example he could have ride in a safe manner, taking extra care in the difficult portions of the track, and avoiding obstacles using extra safe distance, in order for the ANN to learn behaviours that would prevent them accidents; instead, the first author played the game in a fast but risky manner.

The Backpropagation propagation algorithm is used to train an ANN on the training data. The number of iterations is set to 2000000. The learning rate is set to decrease logarithmically from $1 * 10^{-2}$ to $1 * 10^{-5}$. The training is done online at a rate of 2500 iterations a second; this allows the user to observe the ANN as it trains. After training, the average lap time on track L for a computer controlled bike is found to be 2 minutes 30 seconds.

Genetic algorithms are then used to improve the ANN. The trained ANN is mutated to create a population of 20 ANN's. The number of generations has been set to 100, with a population of 20, elitism of 0, a mutation rate of 0.001, a crossover rate of 0.8, a perturbation rate decreasing logarithmically from 0.5 to 0.005, probability to select average crossover over 10 cuts crossover set to 0.2.

The results can be found in the graph below:

From the graph one can see that the lap time is slowly decreasing, but the average lap time in one generation is not always better than the average lap time in the previous generation. This is what was expected with GA. Note that because the perturbation is decreasing logarithmically from 0.5 to a small value, 0.005, and because of the crossover

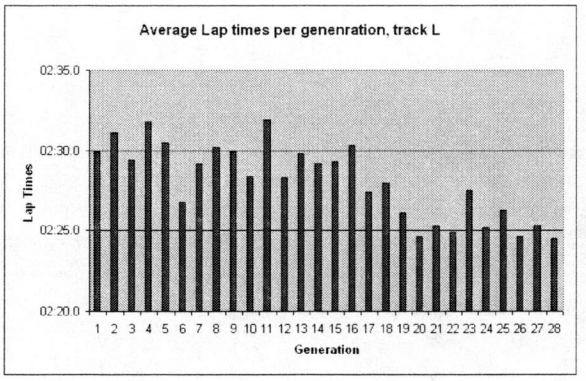

Fig. 2. The average lap time is slowly decreasing with respect to generations, .

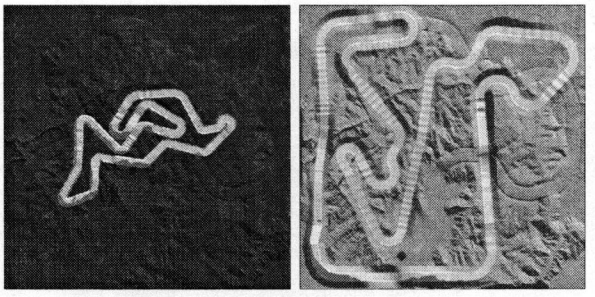

Fig. 3. On the left, track L used to train the ANN. On the right, track O, used to test ANN generalisation property.

techniques used, the individuals in the final generation, at the end of training, are expected to be very similar, and weights between all ANN's can easily be averaged over all individuals to create one ANN representative of all a population.

Finally, we want to check the generalisation property of our ANN's, we present the originally trained ANN (trained using BP), and the optimised ANN (trained using BP and then GA), with track O.

Track O is very different from track L. For example track O features large hills and bumps, not present in track L.

The ANN's are able to generalise. Lap times are 4 minutes 42 seconds for the originally trained ANN, and 4 minutes 34 seconds for the optimised ANN. The ANN's simply seem not to be familiar with the long straight and bumpy portions of track O and are subject to time penalties every time the game engine respawns the bike in the middle of the track. The game engine respawns bikes in the middle of the track if the bikes have been off the track for too long. The velocity of the bikes at respawn is set to be generally less than the velocity of the bikes before respawn; hence a time penalty. For comparison a good human player lap time on this track is 4 minutes 05 seconds.

VIII. Conclusions

Bagging required a lot of processing and memory resources (it was using 10 ANN's per bike instead of only 1),

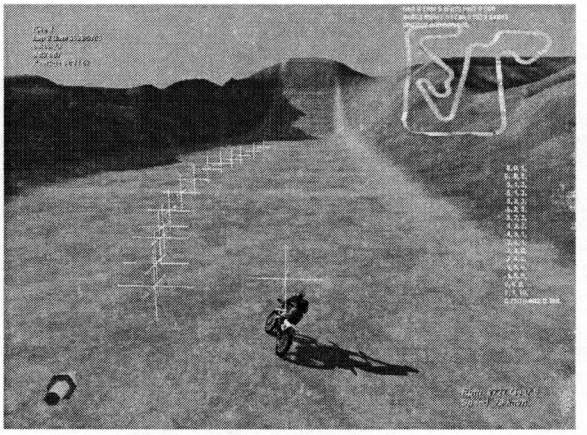

Fig. 4. One large hill in track O.

and still did not prove to give good results. Two techniques, investigated here to improve ANN's training, have proved to give good results; one is boosting, the other one is GA with alternative crossover methods and a population made of mutated already trained ANN's. With evolutionary algorithm, the artificial intelligence can adapt to new track and improve lap times with time; possibly it can eventually perform better than a good human player. Performance so far is nearly as good as that of a good human player. Future work may include optimising the techniques, or investigating new techniques, to reduce training and adaptation time.

References

[1] B. Chaperot and C. Fyfe, "Motocross and artificial neural networks," in *Game Design And Technology Workshop 2005*, 2005.
[2] S. Haykin, *Neural Networks- A Comprehensive Foundation*. Macmillan, 1994.
[3] M. Buckland, "http://www.ai-junkie.com/," Tech. Rep., 2005.
[4] Various, "http://research.microsoft.com/mlp/forza/," Microsoft, Tech. Rep., 2005.
[5] L. Breimen, "Bagging predictors," *Machine Learning*, no. 24, pp. 123–140, 1996.
[6] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," Statistics Dept, Stanford University, Tech. Rep., 1998.
[7] C. Fyfe, "Local vs global models in pong," in *International Conference on Artificial Neural Networks, ICANN2005*, 2005.
[8] L. Breimen, "Using adaptive bagging to debias regressions," Statistics Dept, University of California, Berkeley, Tech. Rep. 547, February 1999.
[9] ——, "Arcing the edge," Statistics Dept, University of California, Berkeley, Tech. Rep. 486, June 1997.
[10] T. Heskes, "Balancing between bagging and bumping," in *Neural Information Processing Sytems, NIPS7*, 1997.
[11] V. Vapnik, *The nature of statistical learning theory*. New York: Springer Verlag, 1995.
[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.
[13] I. Rechenberg, "Evolutionsstrategie," University of Stuttgart, Tech. Rep., 1994.