Motocross and Artificial Neural Networks

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

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

ABSTRACT
In this paper, we investigate training artificial neural networks to ride simulated motorbikes in a new computer game using two different training techniques, Evolutionary Algorithms and the Backpropagation Algorithm. We show that the backpropagation algorithm creates a rider which is faster than that created by the evolutionary algorithm but at the price of requiring a training set created by a human playing the game. Also the evolutionary algorithm has the advantage that it can find solutions which no human has previously found. Both methods create human-like performance in the motocross game.

General Terms
learning, evolutionary algorithms, backpropagation

1. INTRODUCTION
In this paper, we investigate training artificial neural networks to ride simulated motorbikes in a new computer game using two different training techniques, Evolutionary Algorithms and the Backpropagation Algorithm. The use of such methods in control is not new (see e.g.[4] or [9]). There are various interesting aspects in using artificial neural networks (ANN) 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 behaviors which are difficult to express as a set of procedural rules, and make the use of ANN very appropriate.

Using a multilayered perceptrons ANN, we have shown [3] that ANN can learn and perform like a human intelligence (rather than a number-crunching machine). Our main aim is then to have the ANN to perform as well as possible at riding the motorbike, while still retaining the characteristics of a human learning to ride. It is possible in a later stage to cripple the neural network so its performance matches that of a human player for maximum player enjoyment (this has been called artificial stupidity), however one of our aims is to make the computer player learn to ride the bike in a human-like way so employing such tactics is not necessary

Path planning can be performed in an off-line manner such as was investigated in [6]: in such work, there is typically a static environment and the intelligence which may be based on swarm intelligence, evolutionary algorithms, neural networks, artificial immune systems or any other technology which brings intelligence to a task, must find a route through the environment. Such methods do not allow for interaction between players which is one of the features in which we will be interested. Furthermore this method works well for pedestrians but does not take account of the very dynamic and uncertain nature of motorbike riding. While riding a motorbike other very bumpy environment, part of the control is to determine and follow a path, another part is to maintain the balance of the bike.

We know that a multilayered perceptron can learn any function with a countable number of discontinuities and so are confident that we can learn to ride the bike using the backpropagation algorithm, which is the standard method for this type of ANN. However, we also wish to investigate whether the weights of the ANN can be found by evolution. We are interested in both the nature of the learning which takes place (in particular, whether it will appear more human-like than that from the supervised learning method, backpropagation) and also the effectiveness of the resulting ANN i.e. whether one method of learning the parameters outperforms the other.

The rest of this paper is structured as follows: in the next section, we introduce the game, Motocross The Force; we then discuss artificial neural networks and the two methods with which we train the ANNs, backpropagation and evolutionary algorithms. We then perform simulations to compare these methods and finally extend the better of the methods in a variety of manners.

2. THE GAME
There are various interesting aspects in using artificial neural network methods in a motocross game. Because the design of an ANN is motivated by analogy with the brain, and the rationale for their use in the current context is that entities controlled by ANN are expected to behave in a human or animal manner, and these behaviors can add some life and content to the game. The human player has also the

possibility to create new tracks. ANNs 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 have the capability to train and evolve. ANNs 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 screenshots from the game are 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 meter along the track. These way points are used to ensure bikes follow the track and we will talk about 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 meter away from the center of the next way point.

The inputs to the ANN are:

- Position of the bike in way point space.
- Front and right directions of the bike in way point space.
- Velocity of the bike in way point space.
- Height of the ground, for **a** (typically 30) ground samples, in front of the bike, relative to bike height.
- Position of track center lane, for **c** (typically 6) track center lane samples, in front of the bike, in bike space.

3. 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 [3] 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) 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:

$$\begin{aligned} act_i &= \sum_{j=1}^{N} W_{ij} x_j, \forall i \in 1, ..., H \\ h_i &= \frac{1}{1+\exp(-act_i)} \end{aligned}$$

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:

$$\begin{aligned} act_i &= \sum_{j=1}^{H} V_{ij} h_j, \forall i \in 1, ..., O \\ o_i &= \frac{1}{1+\exp(-act_i)} \end{aligned}$$

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.

3.1 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 $(t_i^P - o_i^P)Abs_(t_i^P - o_i^P)$. This has for effect to train the ANN more when the error is large, and allow the ANN to make more decisive decisions, in regard to turning left and right, and accelerating, breaking.

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 and left/right turning 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.

3.2 Evolutionary Algorithms

Genetic algorithms (GAs) became popular after the seminal work of Holland [5] in the 1970s and 80s. His algorithm is usually known as the simple GA now since many of those now using GAs have added bells and whistles [7]. Holland's major breakthrough was to code a particular optimisation problem in a binary string - a string of 0s and 1s. He then created a random population of these strings and evaluated each string in terms of its fitness with respect to solving the problem. Strings which had a greater fitness were given greater chance of reproducing and so there was a greater chance that their chromosomes (strings) would appear in the next generation. Eventually Holland showed that the whole population of strings converged to satisfactory solutions to the problem.

Notice that the population's overall fitness increases as a result of the increase in the number of fit individuals in the population. Notice, however, that there may be just as fit (or even fitter) individuals in the population at time t-1 as there are at time t. In evolution, we only make statements about populations rather than individuals.

We can identify the problem of finding appropriate weights for the MLP as an optimisation problem and have this problem solved using the GA: 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 [7] 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.

For training purposes, the number of generations has been set to 100, with a population of 80 ANN, elitism of 2 (number of the fittest chromosomes being passed directly from the parent population to the child population) which is equal to 2.5%, a mutation rate of 0.1, a crossover rate of 0.7, and a perturbation rate decreasing logarithmically from 0.8 to 0.008.

There is a simple reason why the perturbation is set high at the beginning and low at the end: let us consider an ANN attempting to jump bumps; if for example the bike goes at 30 km/h, then the bike can jump over perhaps one large bump; if the bike goes at 45 km/h, then the bike may be able to jump 2 large bumps at once. However if the bike goes at 35 km/h, then the bike may land on the ascending part of the second bump and is likely to crash. If the perturbation was not set high at the beginning, then an ANN which is successfully jumping one bump would not be able to attempt jumping two bumps at once; any increase in speed would only take it into the crash regime not into the second

safe regime. The perturbation is set low at the end of the training, because as we are approaching from the solution, we don't want to deviate too much from this solution.

Evaluation of fitness of each ANN is carried out for a duration of **vTestTime** seconds maximum, less if the bike is in an unrecoverable situation or if the ANN is evaluated early as not being fit. As an optimisation technique, the intelligence and experience are shared by more than one bike; more than one bike (typically 6) are on the track and the same number of ANN is thus evaluated for fitness at any given time.

4. EXPERIMENTS

The longest track is chosen for experiments. On this track, ANN is presented with many different obstacles and situations.

For backpropagation, we recorded 20 minutes of the first author playing the game to give a training data set of 120000 samples. The target sample is the human's response in terms of turn left/right and accelerate/decelerate, both of which are recorded as floating point numbers between -1 and 1.

The number of iterations is set to 10 million, with, for each iteration, a pattern chosen randomly from the 120000 samples. The learning rate is set to 0.0001. We found that a higher number of iterations does not improve training significantly. A higher learning rate would make the ANN overfit patterns and thus generalisation to new tracks would be lost. A lower learning rate would make the ANN require a higher number of iterations to learn, or it may even not learn at all due to floating point limitations.

Early experiments with Evolutionary Algorithms have shown some interesting behaviors: If **vPassWayPointBonus** is too high (bonus), the ANN tend to learn to instantly crash the bike immediately the bike has been spawned, relying on the fact that the game engine spawns the bike on the track in the right direction after a crash. If **vCrashBonus** is too low (negative bonus, penalty too high), the ANN tend to learn to ride away from the track, where it is safer. If **vTestTime** is too high (say 2 min), then the ANN will tend to control the bike in a very slow and safe manner, in order not to crash and not put itself in an unrecoverable situation. If **vTestTime** is too low, then the ANN will tend to control the bike in a fast but risky manner.

After experimentation, the following values have been determined as suitable:

- **vPassWayPointBonus** = 10
- **vMissedWayPointBonus** = 0
- **vCrashBonus** = 0
- **vFinalDistFromWayPointBonusMultiplier** = -10
- **vTestTime** = 60 sec

The sigmoid function chosen was $m * \tanh(\frac{act}{r})$ with m a multiplier, set to 1.1, and r, the activation response set to 4. $m = 1.1$ enables the sigmoid function to easily output values between -1 and 1. $r = 4$ enables a responsive smooth output from the ANN .

Since the inputs span different ranges, they are all scaled so that on average they lie approximately in the range 0-1000. The ANN output is used as the controls for the motorbike and uses the same controls as a human player. The two controls are acceleration/deceleration, and turn left/right. In fact there is a small difference: instead of taking the ANN output as the acceleration control, the output is taken as a velocity control; this means the ANN does not need to do some derivation work, and optimizes the use of ANN. The derivation work can be done easily through simple computation.

4.1 Results

A good human player can ride a motorbike on one lap round the long track in between 2 minutes 10 seconds and 3 minutes.

An ANN trained using evolutionary algorithms can ride the motorbike around the same lap between 2 minutes 50 seconds and 3 minutes 30 seconds.

An ANN trained using backpropagation algorithm using training data from the good human player, can do lap times between 2 minutes 25 seconds and 3 minutes 15 seconds. The backpropagation method is thus rather better than the evolutionary algorithms.

There has been recent work identifying the most important data samples [8] and on creating pseudo data sets for bagging [1] or boosting [2]. We therefore further wish to investigate 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 part might enable it to concentrate its efforts on the difficult sections of the track and so a training routine has also been developed: 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.

Using the backpropagation algorithm, 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 each training sample now had a probability to be selected for training proportional to the inverse of the error produced the last time the sample was presented to the ANN) this had a positive effect.

Finally, using back propagation algorithm, and the inverted training routine, the ANN could ride the motorbike on one lap round the long track in between 2 minutes 20 seconds and 3 minutes. The performance is very similar to the performance of a human player. Furthermore, the ANN tends

to ride the motorbike in a smooth and human like way, and is also able to recover from unusual and unexpected situations like getting back on track after a accident.

5. CONCLUSION

There are advantages and disadvantages in both training methods, evolutionary algorithm and backpropagation algorithm.

The evolutionary algorithm has the advantage that it can be used to train a neural network, with no recorded training set. The neural network can train and adapt to any new environment or track. It also has the advantage that it is possible at the end of training for the artificial network to find solutions (in our case how to ride the motorbike along the track) a human would not have found. Experiments have shown that the solutions found were original, but were not optimal and often not as good as a good human player solution.

Backpropagation on the other hand gave better results than evolutionary algorithms. The disadvantages of backpropagation are that it requires the creation of training data made from the recording of the game played by good human player. Another disadvantage is that the neural networks trained with back propagation are not trained to deal with unusual cases, like how to recover after an accident, when the bike is not on the track and facing the wrong direction, if the situation has not been met during the creation of the training data, and may not recover as nicely as would a human player or an ANN trained using evolutionary algorithms.

One of the reasons the evolution algorithms did not perform as well as backpropagation may be that the population size and the number of generations were small relative to the huge number of ANN weights to optimize.

Currently we are investigating the use of evolutionary algorithm to select and optimize the set of inputs and parameters used by the ANN, for example, the positions and number of ground samples used as input to the ANN.

There are a number of possible questions arising out of the current work: for example, we used the simple genetic algorithm which is only one possibility from the field of unsupervised exploratory learning algorithms. For example it might be interesting to compare the results achieved here with those from reinforcement learning. It is also far from clear as to which type of game the above results might apply. It seems likely that the results in this paper might be equally valid to other racing games, simulators and, in general, games requiring analogue inputs but this is an empirical question which can only be resolved through future experimentation.

Other future work may be to use evolutionary algorithms to train the ANN, with a much larger population size and number of generations. Obviously this work is constrained by time and/or processing power available.

Other work may include applying swarm intelligence to the problem; for example to have good paths and bad paths and crash points chemically marked along the track, and have the ANN to recognize those chemical markings.

6. REFERENCES

[1] L. Breimen. Bagging predictors. *Machine Learning*, (24):123–140, 1996.

[2] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting,. Technical report, Statistics Dept, Stanford University, 1998.

[3] C. Fyfe. Local vs global models in pong. In *International Conference on Artificial Neural Networks, ICANN2005*, 2005.

[4] S. Haykin. *Neural Networks- A Comprehensive Foundation*. Macmillan, 1994.

[5] J. Holland. Genetic algorithms and adaptation. Technical Report 34, University of Michigan, 1981.

[6] G. Leen and C. Fyfe. An investigation of alternative planning algorithms: Genetic algorithms, artificial immune systems and ant colony optimisation. In *Conference on Computer Games: Design, AI and Education, CGAIDE2004*, 2004.

[7] I. Rechenberg. Evolutionsstrategie. Technical report, University of Stuttgart, 1994.

[8] V. Vapnik. *The nature of statistical learning theory*. Springer Verlag, New York, 1995.

[9] Various. http://www.ai-junkie.com/. Technical report, 2005.

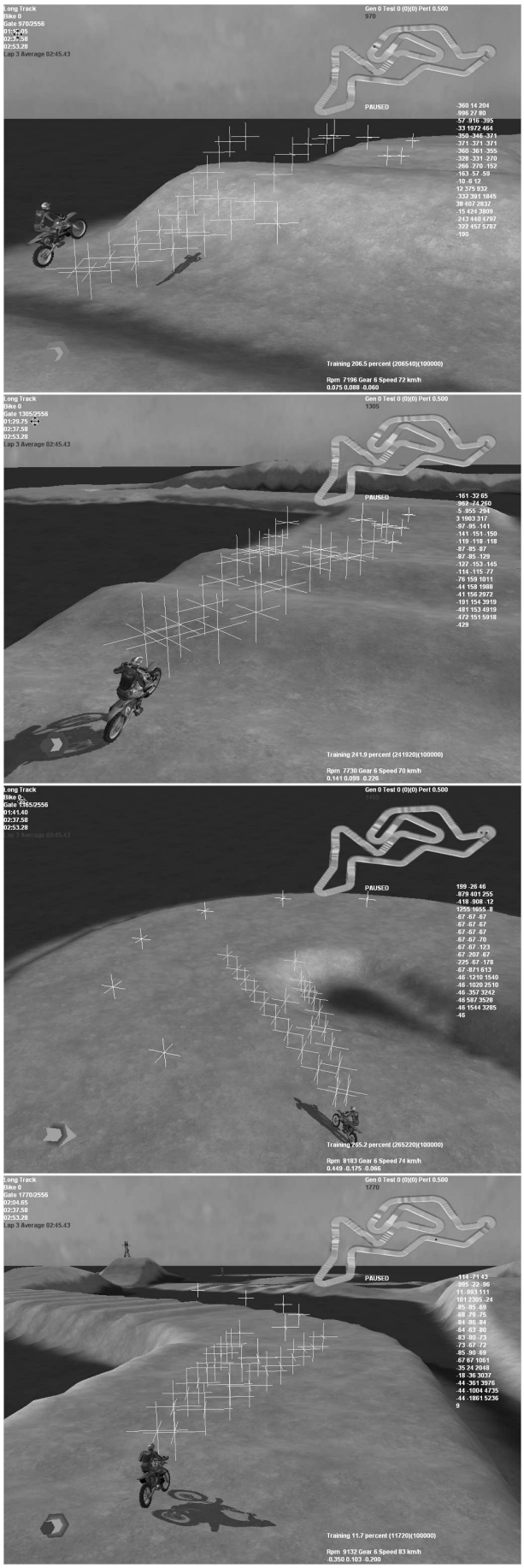

Figure 1: 4 screen shots taken from the game