Virtual table tennis and the design of neural network players

D. d'Aulignac, A. Moschovinos and S. Lucas
Department of Electronic Systems Engineering,
University of Essex,
Colchester CO4 3SQ, UK
email sml@essex.ac.uk

Abstract

This paper discusses the design of a virtual table-tennis environment, and the design of neural network based controllers to play in that environment. The motivation behind the work is to provide an interesting and entertaining forum in which to carry out research on adaptive control and planning problems that stretch the limits of current neural network paradigms.

1 Introduction

Currently there is much interest in neural networks and genetic programming methods for control of real robots. However, to experiment with such things requires expensive hardware that can be time consuming to set up and maintain. An alternative to experimenting with real robots is to experiment with virtual robots. The complexity of designing controllers for such virtual robots depends on the 'physics' of the virtual environment, and the task at hand. This paper describes the design of a virtual table-tennis environment and some initial work on the design of algorithmic and neural network based bat controllers.

Neural networks have been designed or evolved for solution for control problems (e.g. [1, 2] and separately, for strategy games such as OXO [3]. Table tennis (or for that matter, any racquet sport) provides an interesting mix of control problems and strategy problems, making the design of good players extremely challenging. In a recent tournament of table tennis for real robots, rallies were extremely few and far between. By concentrating on the design within a virtual environment, however, we have control over exactly how difficult to make the game, and at least for the initial stage, we can ignore completely problems of visually tracking the ball and accurate actuation of the robot.

The rest of this paper is structured as follows: section 2 describes the simulated environment; section 3 describes the design of an algorithmic controller; section 4 describes the results of some experiments with neural network controllers; section 5 discusses some of issues arising from the work and future possibilities; section 6 concludes.

2 The Simulated Environment

Here we choose a 2-dimensional simulated table tennis game as our environment. This is implemented in an object-oriented style. The system was initially implemented in C++ for both Unix and Windows platforms, but is currently being ported to Java, to exploit Java's platform independence and superior networking capabilities.

The implementation model has three aspects, the logical model, the physical model and the graphical model.

2.1 Logical model

This defines the rules of the game. The responsibilities of the logical model are to detect the end of the game, detect the end of a point, and the winner of the point, and decide which player currently has service. For detecting the end of a point, and the winner of that point, it interacts with the physical model. The logical model contains a state machine for this purpose. The states correspond to the logically distinct states of the game, such as *left_serve, right_serve, left_bat, right_bat, left_table, right_table, left_wins, right_wins etc.* The actual state table is a little more complex than this, since the rule governing a serve is different to the normal run of the game. Each time the physical model detects a collision between the ball and the table (either left or right side), the net, and the left or right bat, a state transition is made depending on the object that the ball collided with.

2.2 Physical Model

The physical model is responsible for applying the laws of physics to the objects in the game. To do this it must update the dynamic (ball and bats) objects and monitor their collisions with each other and with the static objects (table, net, floor and ceiling). After each collision the physical model informs the logical model in order to keep the state machine updated.

2.2.1 The ball

The ball is dynamic yet passive object, which gets hit around according to the laws of physics. The physical model accurately describes most of the features of the real game (except for the missing third dimension) including gravity, air resistance, the effects of spin on ball trajectory and collisions, and the coefficients of friction between bat and ball and between table and ball. These effects can also be switched off in order to provide a simpler game environment if necessary.

2.2.2 The bats

The bats are active dynamic objects. Each bat has an associated bat controller. The bat controller must implement a method called `getForce` that takes as parameters the current bat position vector, the position of the opponent's bat, the position of the ball and a boolean variable to indicate whether or not it is this players turn. Of course, velocity and acceleration information is also useful to the bat controller – but this can be derived from successive values of the position.

2.3 Graphical Model

This is used to display an animated view of a game. For the machine based controllers it is entirely unnecessary, and games can be played much faster without one. However, it is useful to observe the traits of various machine-based players in order to better understand their strengths and weaknesses. Also, it is essential if it is required to allow human players to play against machine-based opponents.

3 An algorithmic controller

Given the above simulated environment, it is possible to make accurate predictions of ball trajectory, make decisions on where to intercept the ball, which shot to play when there, and then make a perfect execution of the chosen shot. It may seem that such a controller should never lose a point, but this is not the case. We limit the force that can be applied to the bat at each time instant, and hence, not all shots are possible, and if a bat can be caught out of position it may even be unable to make contact with the ball.

We have implemented an algorithmic controller based on the above ideas, and as expected, it plays a good game of table tennis – it is difficult for human players to win a point against it.

The main reason for implementing an algorithmic controller was to provide training data for supervised neural networks. To make it more interesting, and to provide more varied training data for the networks, the algorithmic controller makes pseudo-random choices regarding the point at which to intercept the ball, and the shot to execute when there.

4 Neural Network Based Controllers

The main aim was to design neural networks and train them to play a 'good' game. Two approaches were tested. Firstly, a single network was used and trained by the algorithmic controller. For this approach both multilayer perceptron (MLP) and radial basis function (RBF) architectures were tested. For the second approach the task was divided into a small number of neural networks. Each network was trained to do a particular task. Then, they were all combined to integrate a player. For this case an MLP was used.

4.1 Inputs and Outputs

A realistic input vector for a neural network would be the position and velocity of the ball, and the position and velocity of the bat it handles. An output vector would consist of the forces (in x and y coordinates) and the torque that the controller applies to its bat. However, for simplicity at this stage the torque is ignored. We are just interested in moving the bat, while leaving at a fixed angle. Using the algorithmic controller a set of training pattern pairs can be produced. This should include as many representative cases as possible.

4.2 Single Module Newtork

The results of the single module networks at first appeared to be strange. Both the RBF and MLP networks were trained on the training data, and repeatedly tested on the test data until the test-set error reached a minimum. In the case of each network, this was a reasonably small error (of the order of 0.001 mean square error). These learned weights were then hard-wired into a bat controller to play in

an actual game. The RBF marginally outperformed the MLP, but both networks (many different configurations of each one were experimented with) generally performed poorly compared to the algorithmic controller – frequently missing the ball or hitting it way off the table, and on some occasions, even appearing to actively avoid the ball.

The most probable cause of this is that while the network behaves well in the regions of input space which the algorithmic controller inhabits, it has no reason to behave well outside of these regions. As soon as the neural network begins to stray from what the algorithmic controller would have done in a given situation, the problem then accumulates – and the bat is rapidly sent into regions of input space where the algorithmic controller has never explored.

Perhaps a further problem is the pseudo-random behaviour of the bat controller. The neural network models are capable of approximating functional mappings, but the data given to them is not of this nature if we include a random element in the algorithmic controller, since, given identical input conditions, the algorithmic controller can produce different outputs[1].

However, the neural networks still performed significantly better when trained on the random algorithmic controller than when trained on a non-random version. Perhaps the best possibility is to simply generate a large number of random input training vectors together with what the algorithmic controller would output given those inputs, but we have not yet done this.

4.3 Modular neural networks

The second approach to implement a neural network player is to decompose the task into a number of smaller tasks. Each smaller task can then be handled by an independent specialist neural network. With this modular approach, it is of course possible to have different modules based on different paradigms – there is no need for all modules to be neural networks. During development of the system, it is sensible to begin with an algorithmic module for each task. Having checked that this functions well, each module can then be replaced in turn by its neural network alternative. In this way, it is possible to identify which neural network modules are performing well and which ones are performing poorly.

We decompose the problem into three parts: prediction of the intercept point, calculation of intercept vector and movement of bat to achieve the desired intercept.

4.3.1 Calculation of desired intercept point

The first step is to have a network which can predict some point of interception. This involves predicting the position of the intercept and the time at which the ball will be at that position. This is chosen to be the highest point of the trajectory in which the player is allowed to hit the ball. An MLP is used for this stage and actually predicts the point of interception very accurately. Inputs to the network is the position and velocity of the ball when leaving from the opponents bat. The outputs are the x, y and time of the predicted intercept. The training set was produced from the outputs of the algorithmic controller, designed to estimate the highest point of the trajectory. An MLP with a single hidden layer of 22 neurons proved to be good enough for this task.

4.3.2 Calculation of desired intercept velocity

Based again on the outputs of the algorithmic controller, targets can be derived to train an MLP to output what velocity the bat should have at the predicted point. This velocity must be such so as to return a good shot. For this task, neural networks have also been trained successfully. Thus, if we cheat and warp the bat to the desired point with the desired velocity at the correct time, we have a combination of two neural networks which can play as good as the robots (i.e. a game lasting for more than 20 hits!).

4.3.3 Moving the bat

The third network module has the task of moving the bat over successive time intervals in order that at the time of intercept, the bat has the desired position and velocity. Describe the action of this one.

However, another network must be trained so as to apply legal force to move the bat to arrive at the desired point at the correct time with the correct velocity. A neural network for this has been designed but not yet tested. It seems likely that by employing a modular decomposition of the problem we shall be able to develop a highly proficient neural network bat controller – one that plays as well as the original algorithmic controller. This leads on to the next step – evolving superior players.

[1] Unless the random variable is included in the set of inputs to the neural network – though this would further complicate the mapping to be learned.

5 Discussion

We are almost at the stage where a multi-module neural network can play a good game of table tennis within the current environment. The next stage in the work is to take the most successful neural network individuals and apply tournament based evolutionary methods to evolving successively better individuals.

The current implementation of the game has been designed to be an accurate simulation of a real, but 2-dimensional, table tennis environment. There are several ways the set up can be varied. The game can be made simpler by eliminating the effects of air resistance and spin. Alternatively, the parameters which control these can be adjusted to increase their effect, hence making the game more difficult.

Other ways in which the game can be made more difficult are: extend the simulation to a three dimensional environment; make the robot controllers act on some multi-segment robot arm in order to control the bat, rather than applying forces directly to the bat as they do now; limit the amount of computation allowed for each controller at each timestep. This would have the effect of favouring controllers who could not only play a good game, but do it within some bounded amount of computation. Before these are explored, however, there is plenty of scope for improving the performance of the current robot players.

There has been a good deal of interest around the world in our virtual table tennis project, and it is planned to hold an internet-based virtual table tennis tournament. The idea is that people wishing to enter a competitor would submit the code for their controller (having already developed it and tested it on their own machine) – the newly submitted controller would then be pitted against a league of all the best controllers so far submitted, and if sufficiently successful, earn its own place in the league or otherwise be discarded. Over time it would be interesting to see the type of architectures that dominate the tournament, and the kind of games they play. To facilitate this, we are currently porting the simulator to JAVA, and also working out details of a GUI-based neural network controller design system.

6 Conclusions

This paper has described a framework that allows the development and evaluation of robot controllers for a simulated table-tennis environment. The current status of the project is that algorithmic controllers have been designed that play a good game of virtual table tennis. The initial experiment to train a single feed-forward neural network to play virtual table tennis was largely unsuccessful, with the single neural controller struggling to maintain a rally of more than about 2 shots, hence proving no match for the algorithmic controller.

The modular neural networks are far more promising, and a successful bat controller has been constructed using a neural network for prediction of the intercept point, a neural network for the calculation of the intercept velocity, and an algorithmic module for seeing that the bat achieves the desired intercept velocity at the chosen place and time.

Already the project has generated a good deal of interest on the internet. This will hopefully increase when the Java version of the simulator becomes available, which will include a system for the interactive design of robot controllers, and an easy means for people to participate in an internet-based tournament.

Finally, this kind of work provides a natural bridge into the design of real robot game players (i.e. to play table tennis against each other, and/or against humans on a real table tennis table). By developing the details of the robot controller in a virtual environment, much of the design work can be done much more quickly than if having to deal with real robots. Although not a feature of the current implementation, it is of course possible to implement models of real robots within our simulated environment.

Related www sites

For more information on the project and related links, or to download our table tennis simulator, visit our project home-page: `http://giwww.essex.ac.uk/`

References

[1] A. Wieland, "Evolving controls for unstable systems," in *Proceedings of the 1990 Connectionist Models Summer School* (D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, eds.), pp. 91 – 102, San Francisco: Morgan Kaufman, (1990).

[2] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," *Neuro-colt technical report series NC-TR-96-048*, (1996).

[3] D. Fogel, "Using evolutionary programming to create networks that are capable of playing tic-tac-toe," in *Proceedings of IEEE International Conference on Neural Networks*, pp. 875 – 880, San Francisco: IEEE, (1993).