# An Adaptive 'Rock, Scissors and Paper' Player Based on a Tapped Delay Neural Network

Lars Niklasson[1], Henrik Engström[1] and Ulf Johansson[2]

**Abstract: This paper presents an adaptive 'rock, scissors and paper' artificial player. The artificial player is based on an adaptive neural network algorithm. The hypothesis is that human players do not adopt the optimal playing strategy, i.e. to use random moves, and that the artificial player could exploit this and adopt a winning strategy. To test this hypothesis a WAP-based and a web-based version of the artificial player was made available to the general public. A total of about 3000 human players have played against the artificial player, to date. Several different training strategies are evaluated, and the results show that efficient algorithms can be constructed. The best result being 72% won games for the artificial player and 28% won by human players. The paper also identifies future interesting issues for both game developers as well as researchers within Human Computer Interaction.**

## I. Purpose

The purpose of this paper is to show that adaptive techniques (in this case an artificial neural network) can be implemented in automated game players to allow them learn the strategy of the opponent and use this to adopt a winning strategy. Naturally this requires capability to change strategy if the opponent changes strategy.

The game we have chosen here is 'rock, scissors and paper'; a simple game where two players use their hands and simultaneously show the sign for either *rock*, *scissors* or *paper*. The rules of the game is that *rock* beats *scissors*, *scissors* beats *paper* and *paper* beats *rock*. In the simulations presented here a game consists of a number of plays. The winner of a game is the player who first gets to ten won individual plays.

The optimal strategy for a player, who does not know the strategy of its opponent, is to play according to a random strategy, resulting in about 1/3 lost, 1/3 won and 1/3 drawn plays. This means that the optimal strategy here would result in 50% won games (i.e., first to ten won plays) and 50% lost games. However, the hypothesis here is that human players quite often adopt other strategies. Sometimes these strategies might be intended (e.g., the player is trying to win) and sometimes they might be not (e.g., the player is trying to adopt a random strategy but fails). Regardless of reason, the hypothesis is that an adaptive artificial player can discover this an exploit it in order to win.

[1]Department of Computer Science, University of Skövde, P.O. Box 408, 541 28 Skövde, Sweden, Tel. +46 500 44 83 37, Fax +46 500 44 83 99, lars.niklasson@ida.his.se, henrik.engstrom@ida.his.se

[2]Department of Business and Informatics, University of Borås, Sweden, ulf.johansson@hb.se

## II. Method

The method used here, to implement the adaptive capability of the artificial player, is a neural network. The general neural network consists of a number of input units. These send activation in a feed-forward fashion (possibly via some intermediate, or 'hidden' units) to a number of output units using some weighted connections, see figure 1.

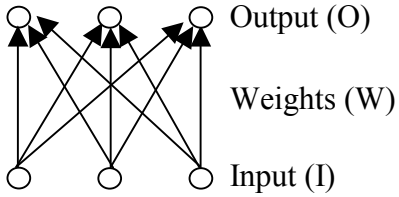


Figure 1: A simplified version of a neural network.

The activation of an output unit ($net_j$) is calculated using the algorithm $net_j = \Sigma(I_i \times W_{ij})$. The output (Oj) of the output unit is then calculated by applying some function on the unit activation, i.e., $O_j = f(net_j)$. The key component of neural networks is the learning algorithm. An error (E) is calculated for each output unit, given a known combination of input (I) and its corresponding target output (T), according to the simple algorithm $E_j = T_j - O_j$. The error is then used to calculate the change of the weights ($\Delta W$), according to the algorithm $\Delta w_{ij} = \eta \times E_j \times I_i$, where $\eta$ is a parameter called a learning rate and can be used to control the amount of learning for individual situations. It can in fact also be an adaptive parameter which is large in the beginning of a training session and lowered the closer to the optimal weight setting the algorithm gets (i.e., the lower the

total error for all units for all training examples gets). The starting point for training is normally a set of randomly generated weights.

This is the simplest form of neural networks. More complex networks include those with an intermediate layer of units, so called hidden units, whose function is to allow more complex functions from input to output. For a more extensive presentation of simple and more complex artificial neural network algorithms, see Rumelhart, Hinton and Williams (1986). The, in this case, important aspect of these algorithms is their 'model freedom', i.e., that they do not require a designer to implement or program a model, instead it is the job of the algorithms to develop a model given some data. New data can always be used to continuously refine the model.

The nature of the problem addressed here is that temporal context is important, i.e., to predict the opponent's next move (in order to counter it) one have to take previous moves (both own and opponent's) into account. This can be implemented by using different artificial neural network algorithms, e.g., Elman's (1990) Simple Recurrent Network, Jordan (1986) nets, or some sort of tapped delay neural network (TDNN), see figure 2.

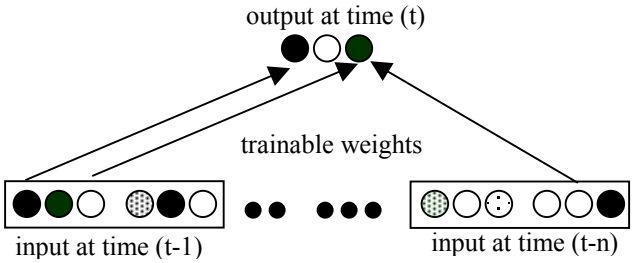


Figure 2: A TDNN. Instead of using only a single input pattern to produce an output pattern, a history of delayed input patterns are used to produce one output.

For a TDNN the main design decision is how many delayed inputs that are needed to capture the context needed to solve the problem at hand. This is often an ad hoc decision, which needs experimentation to be fully understood.

In the simulations present here an automated player based on a TDNN is used. The main motivation for this is its simplicity. TDNNs have previously shown to be useful for solving many different types of sequentially or temporally structured problems, e.g., for text-to-speech conversion (Sejnowski and Rosenberg, 1987) and predicting the impact of advertising (Johansson and Niklasson, 2001).

Five tapped delays are used, each representing the opponent and the automated player's five previous moves (see figure 3). In each time step, both players' moves are represented, by using one unit for each possible move. Therefore, 30 input units are used (five time steps, three possible moves for each of the players). The target is to predict the opponent's next move in order to make the opposite winning move.

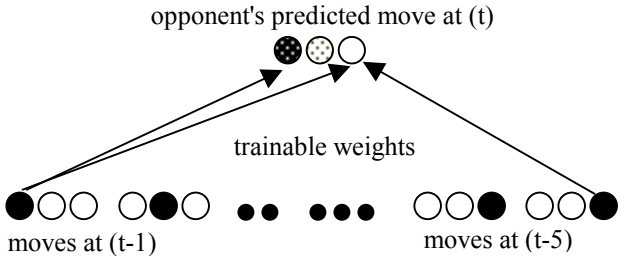


Figure 3: The TDNN used to exploit the sequential nature of the game. Each move for a player is represented by three units (rock, scissors and paper). Here, the five previous moves for both players are used to predict the next move of the opponent.

The output units are linear units, i.e., the output of a unit is the same as the activation for the unit. In each time step the predicted move of the opponent is the move represented by the unit with the highest activation (often referred to as a winner-take-all strategy). In figure 3 the leftmost output unit has the highest activation and its associated move (here *rock*) is therefore chosen as the predicted move of the opponent. The artificial player then selects the opposite move (here *paper*) and displayed at the same time as the human opponent uses the interface and clicks the symbol for the chosen move. The opponent's move is then used as the target output pattern, and the weights are updated according to the neural network training algorithm presented earlier. After this each of the inputs in the tapped delay is shifted to the next delay position in the input buffer, and the last moves of the artificial and human players are stored in the first position in the delay buffer. The network is now ready to predict the next move and repeat the process.

## III. Implementational details

The primary goal with the implementation is to attract a large group of people to play in order to collect empirical data. The game has limited attraction value by itself, which implies that there should be as few obstacles as possible for those willing to play. The first choice was to provide the game as a web application. However, it was hard to attract people to the web-site. This resulted in the development of a WAP-based game-engine. WAP (WAP 1.1) is a technique which enables mobile phones to access the Internet. The number of service providers for WAP browsers is much lower than for traditional web browsers and it was assumed that this makes it easier to get visibility. Also, the game can be attractive for people, e.g. travellers, who want to kill time. It is believed that this is one explanation why people are willing to spend so much time playing the relative simple games built into mobile phones.

A potential drawback with the WAP solution is that players have to pay a per-minute fee when they are on-line. The fee

varies depending on time of day and net service provider. In Sweden the hour rate is typically somewhere between $3-$40. It is here believed that the impact of this cost is neglectable as a game takes less than 3 minutes to play. Actually the cost may have a positive effect in that it prevents players to spend unreasonable amount of time trying to win a game.

A WAP browser can be seen as a reduced WWW browser with no advanced scripting functionality, applets or plug-ins. The screen is normally monochrome with approximately 100*60 pixels. This implies that a game implementation cannot put any application logic in clients. The graphic capabilities of the clients are also a limiting factor. The most commonly supported WAP-version in mobile phones is WAP 1.1. The mobile phone clients communicate with stationary servers through a gateway which is typically managed by the net service provider. Information is sent to the clients formatted in WML (WML 1.1) which can be seen as a specialised subset of HTML. Figure 4 illustrates components of the WAP programming model.

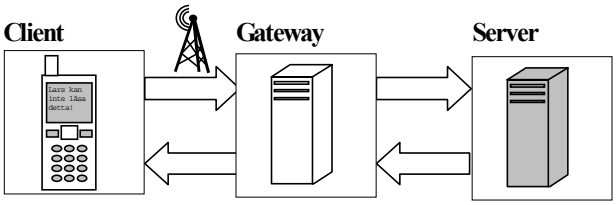


Figure 4. Components of the WAP programming model (From WAP 1.1).

The primary design goals for the application have been to give players quick interaction and flexible control. An initial exploration of the WAP technology showed that it was relatively immature. Different clients and gateways gave notably different result. Therefore the choice here was to use the simplest technique with a minimum of functionality in the clients. This means that everything has to be handled by the server. A dedicated game server in Java, which interacts with clients by sending simple WML-pages, was developed. The game-server is main-memory based which makes it relatively fast. Each new player is assigned a unique session with own weights.

A new session is initiated whenever a client is making a request to the following WAP address[1]:

`http://www.ida.his.se:2002`

After the weights have been initialised and the computer has chosen its first move, the player will be asked to select a move. Figure 5 shows an example on how this may look in a WAP browser.

[1] An HTML version is available at: http://www.ida.his.se/ida/~henrik/rsp/

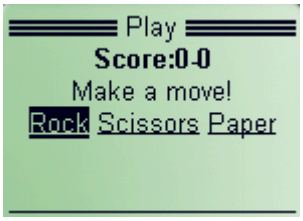


Figure 5: An example on how a player selects a move in a WAP browser.

When the player selects one of the links a request is sent to the server which will memorise the move and use it for the training of the network and to update the score. The response sent back to the client (figure 6) contains the result of the play together with the player's moves.


Figure 6: An example on how the result is presented.

When a game is completed the score of the game is recorded in the server and the player is offered a new game. If a player is inactive for 10 minutes the session is removed which means it will no longer be possible to identify this player or reuse the associated weights.

It is important to note that the server predicts the move of the opponent (which will be used to select a winning move) before the human player makes his/her move. It would obviously be possible for the server to cheat. An alternative solution is to include the computer move in the first page (figure 5) and hide the result in a card which is shown whenever the player makes his/her move. This, however, requires more advanced functions of WML (using decks of cards) which does not work properly in all browsers. Moreover, it would make it possible for human players to cheat which could introduce errors into the results.

## IV. Results for some initial simulations

Some initial simulations uncovered a problem with the learning part of the approach. The artificial player could be played to a position where the human could exploit the situation. The problem was that the artificial player always learned, irrespectively if it was winning or loosing. This meant that it was enforcing a winning move even stronger, which made its future moves predictable.

Instead a new approach was tested. This was to use different learning rates for different situations. The learning rate was set to 0.01 if the artificial player won the move, it was set to

0.2 if the move was a draw and 0.5 if the artificial player lost. The hypothesis was that it was little point in changing a winning strategy. After this change, the network became very hard to beat if one really tried.

The potential of the adaptive player to detect different strategies was tested against some pre-defined strategies. The adaptive player was tested against a random player with same distribution over the different moves and a random player with a biased distribution over the different moves. Neither of these take into account the temporal context of the game, i.e., the previous moves of the players. Therefore a simulation was conducted were the adaptive player was played against a strategy always using the move of the adaptive player at time (t) as the opposing move at time (t+1). The idea was to investigate if and how fast the adaptive player could discover this and adopt a winning strategy. To evaluate how much contextual data that was needed to capture this, two data sets were used; one where a play was decided to continue until any of the two players had won 10 individual plays, and one which continued until any of them had won 20 plays.

The artificial player was also made available on the Internet and WAP. To date 2887 games have played against the automated player. These games have been divided into two different sets. The first set contains 2526 games and the second 362. The reason for this division is a change of training strategy of the artificial player.

Using the set-up suggested here, a ten-wins game appeared to be too short, when playing against a human, to give the network an advantage for fully exploiting the strategy of the opponent. The artificial player therefore wins about 50% of these games, which is as good (or bad) as a random strategy. However, an option to play a second successive game is also possible. This means that the artificial can use the trained network from the first session as the starting point for the second game, rather than a new set of random weights. The result for the second successive game is always significantly higher compared to the first game.

### *A. Results against a random player*

To analyse the behaviour of the artificial player, a set of simulations were conducted where it played against other automated players. The first experiment was to play against a truly random player. As expected, the artificial player cannot utilize this[2]. The random player will win 50% of the games also when several successive games are played using the same network.
In a variation of the random player, the same sequence of random moves was iterated. The result shows that an artificial player is able to utilise these patterns, even with relatively long cycles. With 20 moves in the cycle, the artificial player won on average 57% of the first games and 89% of the second games. It is notable that the performance is highly dependant on the generated sequence.

[2] Actually, as the opponent's moves are based on a psuedo-random distribution it would be possible for an opponent to utilize this.

### *B. Results against a biased random player*

The second experiment was to play against a random player which picked its move from a skew distribution where *paper* was picked with a different probability than rock and scissors (which were picked with equal probability). Figure 7 shows the percentage wins for the artificial player as a function of the probability for *paper*.

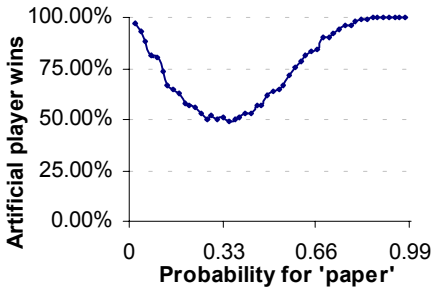


Figure 7: The performance of the artificial player against a skew random player.

Each point on the curve in figure 7 corresponds to 1000 non-successive games. With successive games, the artificial player continues to improve and the result gets slightly better.

### *C. Results against the mimic-approach*

In all experiments up to now the automated opponent has been ignoring the actions of the artificial player. A mimic player was developed which, in contrast to the other automated players, uses the opponent's moves as input. The mimic player selects a move under the assumption that the opponent will repeat its previous move. This strategy showed to be trivial for the artificial player to defeat. It won 1000 out of 1000 games during experiments.

### *D. Results against human players*

The results for the 2526 games of the first set was that the artificial player won 49% of the 2194 games where the starting point for the artificial player was the set of random weights. In 332 cases the human player decided to play a second successive game. This increase in training data meant that the artificial player won 66% of these second games.

The result so far shows that the network can exploit the apparent strategy of the human players (given sufficient amount of background data). However, some design decisions could possibly be changed to further improve the results. Therefore a second set of simulations were conducted.

## V. Results for an extended approach

The hypothesis was that the starting point for the network was not the optimal one. To this point the starting point was to use randomly generated weights. Which naturally meant that the first predicted move was randomly chosen. However, this might not lead to generating sets of randomly chosen moves. It could mean that biased sequences of moves were generated.

In order to test this hypothesis a change in the playing strategy was made. Instead of only starting from a set of random weights, the artificial player was played against another automated player. The idea being that a random play would develop, which would be a very good starting point when playing against a human player. The number of artificial against random player pre-game moves was decided to 100.

### *A. Results against a random player*

As expected, there was no improvement in performance when the artificial player was faced with a random player. The explanation to this is that the random player is not affected by the behaviour of the opponent.

### *B. Results against a biased random player*

Again, there is no improvement in performance.

### *C. Results against the mimic-approach*

The artificial player is superior over the mimic player also in the extended version.

### *D. Results against human players*

After this change the artificial player was once again made available on the web and WAP. Over a period of time 361 new games were played. After the extended training approach, the artificial now won about 55% of the first 10 winning-moves sets. As in the first simulation, those players who went on to play a second successive game, lost to an even larger degree; namely 72% of the games won by the artificial player and only 28% won by the human player. It deserves however, to be pointed out that the number of games here is only 29 which means that the result should be considered somewhat uncertain as yet.

## VI. Discussion

The hypothesis for the work presented here was that an adaptive algorithm can exploit the strategy of a human player playing an, in essence random, game. The best result so far is one where the artificial player wins about 55% of the games played against human players, with an even better performance (73% won games) when a player plays a second successive game (giving the network more training data).

Several extensions to this basic set-up are possible. The simple type of neural network used here allows only simple linear associations between the input and output. A natural extension would be to test a network with hidden units, allowing also non-linear associations between inputs and outputs. One potential problem is however the limited amount of data, which supports the simple solution used here. Other extensions include using longer tapped delays, allowing modelling of more complex mappings. However, this also suffers from the problem with limited data for training. Instead one could test to use recurrent networks and let them learn the length of the 'important' delay. This naturally could be different for different human players.

This sort of simulations can also be of interest for researchers within Human Computer Interaction (HCI). It would be very interesting to see if there is any difference in the human strategy when playing against a human compared to an artificial system. This could easily be explored by comparing different sets of human players, some playing 'directly' against the artificial system, and some playing against a human using the moves suggested by the artificial system.

For the community interested in developing computer games, the adaptive capability appears invaluable. The fact that about 3000 individuals pay good money to use a WAP phone and interact with an extremely simple interface to play an essentially random game should alert the game designers that the adaptive capability is needed in computer games of the 21st century.

## VII. Acknowledgements

This research is supported by the foundation of knowledge and competence development, Sweden, under contract 1507/97.

## VIII. References

[1] Rumelhart, D. E., Hinton, G. E. and Williams, R. J., Learning Internal Representations by Error Propagation, *Parallel Distributed Processing -volume 1*, MIT Press, 1986, pp 318 - 362.

[2] Jordan, M. I., *Serial Order: A parallel distributed processing approach*, Tech. Report No 8604, University if California, Institute for Cognitive Science, 1986.

[3] Elman, J, Finding Structure in Time, *Cognitive Science*, 14, 1990, pp 179 - 221.

[4] Senjowski, T. J. and Rosenberg, C.R., Parallel Networks that learn to pronounce English text, *Complex Systems*, 1, 1987, pp 145-168.

[5] Johansson, U. and Niklasson, L., Predicting the impact of advertising - a neural network approach*, The International Joint Conference on Neural Networks*, 2001

[6] WAP 1.1, "Wireless Application Protocol Architecture Specification", 1998, http://www1.wapforum.org/tech/documents/SPEC-WAPArch-19980430.pdf

[7] WML 1.1, "Wireless Application Protocol Wireless Markup Language Specification Version 1.1", http://www1.wapforum.org/tech/documents/SPEC-WML-19990616.pdf