Evolution of a Learning and Anticipating Decision System

Alexandra Mark, Bernhard Sendhoff, Heiko Wersing

2004

Preprint:

This is an accepted article published in Third Workshop on SelfOrganization of AdaptiVE Behavior (SOAVE 2004) Ilmenau. The final authenticated version is available online at: https://doi.org/[DOI not available]

Published in: SOAVE 2004 3rd Workshop on SelfOrganization of AdaptiVE Behavior, pp. 271-281

Evolution of a Learning and Anticipating Decision System

Alexandra Mark, Bernhard Sendhoff, and Heiko Wersing
Honda Research Institute Europe
Carl-Legien Strasse 30
63073 Offenbach/M, Germany
{alexandra.mark, bs, heiko.wersing}@honda-ri.de

Abstract

We describe an adaptive decision making architecture which is applied to competitive games. The task is to learn online a model of the current opponent strategy which is used to predict the next opponent actions in order to find an appropriate counter-strategy. We show this for the iterated prisoner's dilemma and rock-paper-scissors. In comparison with three other methods against different typical game strategies as opponents, our system performs best in most cases. In some experiments, the best prediction accuracy did not lead to the best payoff. This phenomenon is discussed in more detail.

1 Introduction

Our motivation is to develop a general decision making system which can solve different strategic problems using methods like evolutionary computation and learning. We focus on problems that fulfill certain conditions such as a fixed number of possible actions, an existing payoff matrix or fitness function, and discrete time steps, but they do not have to satisfy the Markov property and are not restricted to games – our first field of applications. There are for example navigation tasks which fulfill these constraints and might be potential future applications for our system. According to [8], humans consider different alternatives, learn and adapt their strategies, when solving decision making problems. Thus they internally play through different potential outcomes, using various models of their environment and considering different own actions. Evolutionary methods are predestinated to solve problems where a reservoir of such alternative potential solutions has to be generated. A good example for this is given in [3]. Darwen and Yao develop an evolutionary system which uses a genetic algorithm to create offline a pool of specialist strategies for game playing. These strategies are used as potential opponent-models for a hypothetical game to evaluate the outcome of different actions.

We have implemented an adaptive decision making architecture which controls the behavior of a player in a competitive game. To achieve this task it is sensible to build an internal model of the opponent players' strategies. There are many different methods to build such an internal model, as for example lookup-tables [3], finite automatons [2], influence diagrams [11], nested recursive models [13], classifier systems [7], or (recurrent) neural networks [12]. These internal models can be used to predict future opponent actions and to find suitable counter-strategies. According to [10] (p. 1593) an "anticipatory capacity is crucial for deciding between alternative courses of action". This

view is supported by [1]. The authors in [4] also use an anticipatory mechanism including internal simulation of hypothetical actions, but they are focusing on the prediction of sensory consequences in a sensorimotor navigation task while we are primarily interested in strategic problems.

When humans interact in competitive situations, their predictions of each other are unreliable and the model which each one makes of the other depends on the model which the other one makes of oneself. In such a nested co-learning system players are often trapped by a pair of false models. For example in the iterated prisoner's dilemma (IPD) small prediction networks often over-simplify the opponent and wrongly recognize the opponent strategy as all-defection [5]. The best counter-strategy against all-defection is all-defection and thus the game will actually end with mutual defection, if the opponent correctly recognizes the all-defection strategy. Then the prediction ability of this small prediction network might misleadingly seem very good, because it correctly predicted the all-defection strategy. In our results we will also see, that the strategies with the best correct prediction rates of their opponent are not always the most successful ones.

In this paper, we present a decision making system (DMS) which is applied to two different games: the iterated prisoner's dilemma and rock-paper-scissors. Our strategies are represented by neural networks. We employ a genetic algorithm to optimize the structure of the networks to allow a good strategy coding. Additionally a learning mechanism is applied to the networks in order to realize an online adaptation to the current opponent. Thus we couple evolution and learning to benefit from the advantages of both methods: The genetic algorithm generates a pool of potential network strategies which might be well suited for learning, and the learning process takes these generated nets and trains them to predict the future actions of the opponent. This pool of individuals from where the best individual is selected as "imitator" is called "imitator-pool" in the following.

In section 2, we describe the structure of our DMS as well as the mechanisms it employs. Evaluation results against different strategies and the comparison with other methods are presented in section 3, followed by a discussion concerning prediction accuracy. We conclude in section 4.

2 Method

2.1 Games

In this paper, the task of our decision making system is to decide on the strategy of a player in 2-player iterated prisoner's dilemma (2IPD) and rock-paper-scissors (RPS). The opponent is given from outside, e.g. by a fixed strategy. In both games two players have to choose one action simultaneously in each round. In 2IPD each player has to decide whether he cooperates ("C") or defects ("D"). In the 2-person zero-sum game RPS the actions rock ("R"), paper ("P"), and scissors ("S") can be chosen. In a variant there exists one more action called well ("W"). Paper covers rock (thus the player who has chosen paper wins), rock crushes scissors (rock wins), and scissors cuts paper (scissors wins). If both players make the same choice, they tie with each other. If well is added, then well wins against rock and scissors (because they can fall into it), well loses against paper and ties against well. The corresponding payoff matrices for both games, which determine the score for each player in one round, are shown in Figure 1. The player which plays the action given in the line of the matrix gets the given payoff against a player which plays the action denoted in the column. These payoffs are summed up over all rounds of one game. The player with the maximal sum at the game end has won.

2.2 Basic Algorithm

We have taken the following method by Darwen and Yao (see [3]; in the following called D/Y) as starting point for our investigations: A Genetic Algorithm (GA) is used to develop a population of

IPD strategies. The last generation is kept as a diverse repertoire of specialized strategies. After the game has started, a gating algorithm searches in this repertoire for the strategies which resemble most the current opponent. The corresponding individuals are called “imitators”. Then all strategies of the repertoire are tested against the selected imitators in a game over 4 rounds, starting with the current game history as input. This corresponds to a prediction of the future behavior of the opponent and a test of potential counter-strategies against it. The strategies which score best against the imitators vote for the next action to be taken in the current situation against the actual opponent.

The main differences between D/Y and our DMS are:

- Our strategies in the repertoire are represented by neural networks instead of lookup tables. This allows a more flexible coding of more sophisticated strategies.
- Our GA optimizes which part of the game history is used as input for each neural net. The GA continues online during the whole game.
- The strategies which are represented by neural nets learn online to imitate the actual opponent.
- We do not search in the strategy repertoire for potential counter-strategies against the current opponent, but we directly test all possible actions against the selected imitator. This assures that the optimal counter-action can be found, when the prediction by the imitator is correct.

	C	D
C	3	0
D	5	1

	R	P	S	W
R	0	-1	1	-1
P	1	0	-1	1
S	-1	1	0	-1
W	1	-1	1	0

Figure 1. Payoff matrix for IPD (left) and RPS (right)

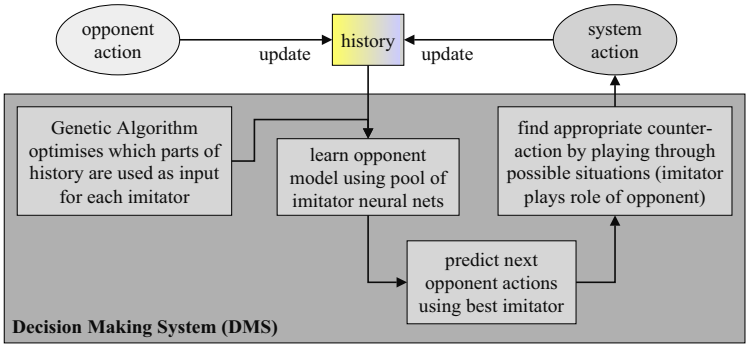


Figure 2. Decision Making System (DMS)

Our basic algorithm works in the following way:

1. create imitator-pool with random chromosomes
2. create a neural net for each imitator with random weights
3. repeat until end of this game (given number of rounds)
   (a) compute system action
      i. in first few rounds randomly
      ii. else
         A. select from imitator-pool best imitator of opponent in past 16 rounds of game
         B. use selected imitator to predict opponent behavior 4 rounds into the future
         C. test all possible actions against the predicted opponent actions for 4 rounds into the future and select the best reaction as the next action of the DMS
   (b) get action of opponent
   (c) compute payoff for this round
   (d) update histories

(e) all *gatRounds* rounds: optimize imitator-pool by GA and RPROP-learning

gatRounds is a random number between a given minimum (default: 3 rounds) and maximum (default: 5 rounds) which determines after how many rounds the system learns. Figure 2 visualizes the important algorithmic steps. Details are explained in the following sections.

2.3 Genetic Algorithm

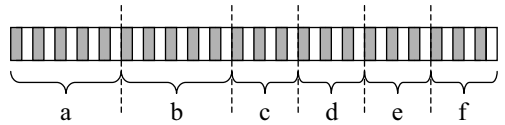
The GA works on the chromosomes of the individuals in the imitator-pool, which are described in section 2.4. One GA-iteration works as follows:

1. delete worst $popSize/2$ individuals from parents; other $popSize/2$ parents are kept
2. to keep population size constant, $popSize/2$ offspring individuals are created as follows
   (a) one offspring gets random chromosome
   (b) create other $popSize/2 - 1$ offspring chromosomes:
      i. with 60% probability in the following way: select one parent fitness-proportionally and the other parent randomly; then offspring is created by 2-point-crossover
      ii. with 40% probability in the following way: select one parent fitness proportionally and use it directly as offspring
3. for each offspring
   (a) mutate chromosome (flip probability 20%)
   (b) create neural net with random weights (number of input neurons in chromosome)
   (c) each neural net learns with RPROP algorithm from previous game history to imitate opponent (some learn parameters are given in corresponding chromosome)
4. add new offspring to new parents population
5. fitness computation of population as described in section 2.6. Considered are
   (a) (possibly weighted) number of matches with opponent actions in the previous rounds
   (b) if the individual has already given a correct imitator-prediction once before
   (c) a small penalty for each used history window

In the experiments shown at the end of this paper we used a population size of 50. In each optimization step (see section 2.2) 10 generations of the genetic algorithm are executed and the population is stored as a new optimized imitator-pool. The learning of the corresponding neural nets is described in section 2.5.

2.4 Strategy Representation

In each round the actions of both players are stored in a game history – one for each player. The actions are coded as consecutive integers: “0” = defect and “1” = cooperate for IPD, and “0” = rock, “1” = paper, “2” = scissors, “3” = well for RPS. The index of the history vector denotes the round of the game. Each individual (or strategy) is represented by a neural network with its weights and a chromosome which codes additional parameters. The nets are fully connected feed forward neural nets with one hidden layer of 10 neurons. Each net gets as input a part of the current game history and gives as output an action. The GA optimizes which part of the history is used as input (see below), thus the number of input neurons of the nets differ.



with
a: windows for own history
b: windows for opponent‘s history
c: number of learn steps from learn history (*numLearnSteps1*)
d: number of learn steps from game history (*numLearnSteps2*)
e: number of learn rounds (*maxLearnRounds*)
f: further learning parameter (*learnBackHist*)

Figure 3. Coding of the binary chromosome

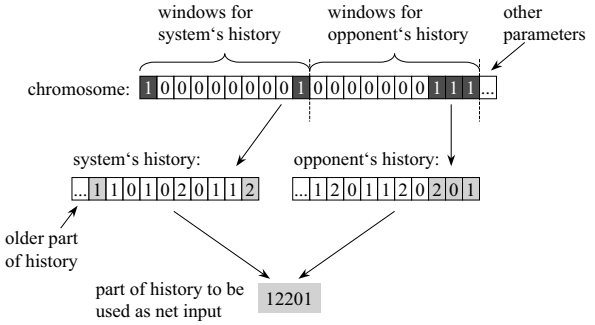


Figure 4. History windows code which parts of the history are used as input for the neural net that predicts the next action of the opponent

The binary chromosome codes several parameters (see Figure 3). The first 10 bits code the windows to look on the system history. If the bit is “1”, then the corresponding position of the history is used, but not if it is “0”. The first bit corresponds to the oldest part of the history (the action before 10 rounds) and the 10-th bit corresponds to the most recent action. The same applies for the next 10 bits which code the windows for the history of the opponent (see Figure 4). “10” is the default value of a parameter which determines how many windows can be maximally used.

The next four values (c–f in Figure 3) are integer values. *numLearnSteps1*, *numLearnSteps2*, and *maxLearnRounds* lie between 0 and 50, whereas *learnBackHist* can take values between 0 and 100. These four parameters are needed for the learning process and are described in section 2.5. For each value 6 bits are Gray coded and rounded to an integer value in the given interval. Thus the length of the chromosome is $2 \cdot 10 + 4 \cdot 6 = 44$.

The length of the lookup-table-type chromosome of [3] increases fast with a rising memory length of the coded strategies. For a memory length (m) of 4 the chromosome length is already 264 ($2^{2m} + 2m$). This is the longest memory length which has been considered in Darwen’s and Yao’s paper. Since we are using neural nets in addition to the chromosome to code our strategies, our chromosomes are much shorter and independent of the memory length.

2.5 Learning

We use fully connected feed forward neural nets with one hidden layer of 10 neurons. Since the GA optimizes which part of the history is used as input, the number of input neurons depends on the number of history windows (h) which are coded in the chromosome. Let *numActions* denote the number of possible actions for each player. Thus the input (which is a part of the game history) consists of h integer values between 0 and $numActions - 1$. Each input value is 0-1-coded and given into $numActions$ input neurons in the following way: For input value 0 the first neuron gets input 1 and the neurons 2 until numActions get input 0. For input value 1, the second neuron gets input 1, and the others get 0. See also Figure 5 for clarity. Thus the number of input neurons is computed as $h \cdot numActions$.

The input neurons propagate the given input directly to the next layer. The hidden neurons and the output neurons have a sigmoid activation function $z_i = 1/(1+exp(-a))$, with a being the propagated result of the previous neurons: $a = \sum_j w_{ij} z_j + \Theta_i$. Θ_i is a bias term between -1 and 1. w_{ij} is the real-valued weight $\in [-1, 1]$ of the connection from neuron j to neuron i. When a new net is created, all weights and bias values are initialized with uniformly distributed random values $\in [-0.1, 0.1]$. Each

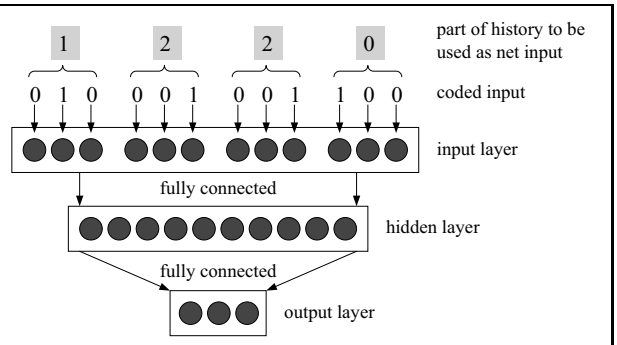


Figure 5. Neural network with example input coded for $numActions = 3$

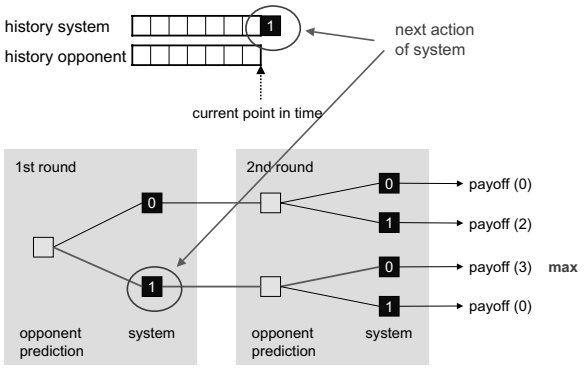


Figure 6. Action selection tree

network has *numActions* output neurons – one for each possible action. The action corresponding to the neuron with the maximal output value determines the action of the corresponding individual (winner takes all). If during the game a net should be used to compute an action, but the available game history is shorter than needed according to the history windows, a random action is returned.

The RPROP (resilient backpropagation, see [9]) rule is applied as learning mechanism to adapt the real-valued weights. The learning data are taken from the past game history: One random position between the current position and up to *learnBackHist* rounds back into the past is determined in the history. The target output for the net is the action of the opponent in the round after the selected random position. Thus the input for the learning net is read from the history (with the random position as newest available history data) according to the history windows which are coded in the corresponding chromosome, and the error is computed using the difference between net output and target. The parameter *learnBackHist* is coded in the chromosome, as well as the number of learn steps *numLearnSteps2* for each learning process and the maximal number of rounds of the game in which the net can learn (*maxLearnRounds*). Thus all *gatRounds* rounds each imitator net does *numLearnSteps2* steps of RPROP learning until the total number of learn rounds (*maxLearnRounds*) is reached. Then the weights of the net are "frozen". This allows to keep strategies as they are in the pool.

numLearnSteps1 was intended to determine the number of steps to be learned from an artificially created history which should be given for initial learning before the start of the real game. The idea was to provide with this learn history some typical strategy mixture. Results showed however, that the online learning is so fast, that no beforehand learning is needed. Thus in the experiments at the end of this paper, the value of *numLearnSteps1* was ignored and no offline learning was done.

2.6 Imitator Selection

In each round of the game the best imitator is selected from the imitator-pool in order to predict the next action of the opponent. The following items are considered to find the best imitator. The values add up to a reward, which is also used as fitness for the genetic algorithm.

- A parameter (*numPastRounds*, default: 16) specifies the number of previous rounds of the game history in which each individual is compared with the real opponent. For each round in which an individual selects the same action as did the opponent in this situation, the individual gets 1 point. Here the individual gets the same history as input which was available to the opponent at this time step.

- Each individual which has been selected once as imitator and has predicted the correct action, gets a positive additive value (+1), which contributes to the reward (r), but decays slightly with each round *t* of the game. Thus with *decay* being a parameter between 0 and 1 (default: 0.95):

$$r(t+1) = \begin{cases} (r(t)+1) \cdot decay & , \text{ if correct pred.} \\ r(t) \cdot decay & , \text{ else} \end{cases}$$

- For each used history position the reward is reduced by a small value (default: 0.2).

The individual with the highest reward is selected as imitator. In the rare case of several individuals having the maximal sum, the imitator is selected randomly among these. The parameter default values have been chosen after some test experiments.

2.7 Reaction to Imitator Action

The following mechanism for finding the best reaction to the predicted opponent action has been implemented: All possible actions which can be played starting with the current game history are tested for the next *numNext* rounds into the future in a hypothetical game. Here the part of the opponent is simulated using the imitator and the system player tests all possible actions it could take. The system action of the first tested round which led to the maximal payoff after all *numNext* rounds is taken as the next action of the system in the actual game (see Figure 6). Obviously this method can only be employed in those cases where the number of possible actions is rather small and all are known. Thus this approach has to be enhanced to work with more general problems.

In [3] all individual strategies in the population from where the imitators are selected also serve as pool from where answer strategies are selected. This method has the drawback, that the reaction-strategies are limited to a small pool of given strategies. So a good prediction of the next actions can possibly not be exploited to full extent, because no suitable counter-strategy is available.

3 Results

For evaluation we compared our DMS with three other methods: “D/Y”, “linear prediction”, and “single neural net”. D/Y is our re-implementation of the system described in [3]. In the linear prediction method, we used a classical linear predictor of order 3 to predict the next action of the opponent, with the game histories of both players as input. In this case the actions have been rounded to the allowed values. In the single neural net method we used a neural net similar to those in the DMS, but with a fixed number of input neurons: the previous 5 actions of both players’ histories are fed into the net which learns online, using the RPROP-algorithm (as in the DMS), to predict the next action of the opponent. For the latter two methods, the counter-action is computed in the same way as in the DMS (see Figure 6), but instead of an imitator prediction, the prediction of the single available linear predictor respectively neural net is used. We had our DMS play 10 runs of 1000 rounds against each of the default strategies of the prisoner’s dilemma competition at CEC’04 [6]:

Figure 7 shows the results. On the right side the percentage of correct predictions of the next opponent action is plotted. The D/Y method is not included here, because no explicit prediction is used. They select a number of imitator strategies instead, which are used to find a counter-action.

The corresponding Figure 8 shows average (avg), minimal (min), and maximal (max) payoffs as well as standard deviations (std) and percentages of correct opponent predictions (pred) of the different methods. The DMS gets sensible results against all strategies. It plays (nearly) only defection against RAND, ALLD, ALLC, and GRIM. This is the best, it can do, when not knowing the opponent strategy but having to explore it online during the game. Against TFT and STFT mostly cooperation

RAND: makes random moves
ALLD: always defects
ALLC: always cooperates
GRIM: starts with cooperation and plays ALLD after first defection of opponent
TFT: tit-for-tat starts with cooperation and repeats last move of opponent
STFT: suspicious TFT: like TFT, but defects in first move
TFTT: tit-for-two-tats: like TFT, but defects only after two consecutive opponent defections
Pavlov: starts with random move and repeats its last move, if it has brought many points (two higher payoff values in Figure 1), otherwise makes other move

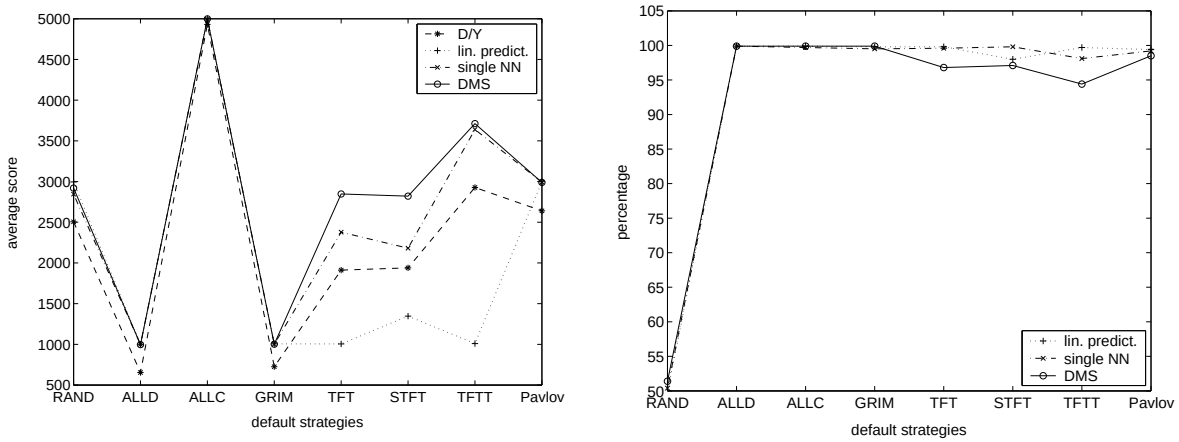


Figure 7. Average score (left) and percentage of correct opponent predictions (right) of different methods against CEC'04 IPD default strategies (10 runs of 1000 rounds for each method; payoff matrix see Figure 1). Standard deviations are given in Figure 8

is played. Against TFTT after an initial phase of mostly defection, the DMS repeats most time the pattern "1010". Against Pavlov the DMS either played many rounds of defection or some irregular pattern of "0" and "1". Compared to the other methods, our DMS belongs always to the best methods and often gets *the* best score. Against the simple strategies ALLD, ALLC, and GRIM, the DMS has no chance to outperform the other strategies at large. The games of the DMS against TFT and STFT have never ended with mutual defection, which has led to the low scores of the other methods. The very high standard deviations of many methods against TFT and variants occur because some of the games have converged to mutual cooperation (high score), and some to mutual defection (low score). The D/Y method shows a relatively high standard deviation against all default strategies. This can probably be explained by the method itself: the strategy reservoir is created offline before the game starts, optimized by a GA which uses for evaluation only games between strategies of the reservoir itself. Thus the population might converge to very special strategies, e.g. rather cooperative ones, which cannot play well against all unseen test strategies. In [3] Darwen and Yao do not measure the diversity of their created reservoir to prove this. In the IPD results shown in Figure 8, our DMS was better than the single learning neural net on average, but not in the maximal values for the TFT variants. Closer analysis showed that both methods usually find good solutions very fast, but the DMS tries some defective moves in between every now and then. Thereby the DMS games never ended with mutual defection – in this respect the DMS is more robust. According to [5] humans' prediction learning does not converge as fast as typical neural network learning, thus allowing more exploration. In our experiments the DMS also behaved more explorative than the single neural net.

We also tested four sophisticated methods playing IPD and RPS against the DMS (see Figure 9).

		D/Y	lin. pred.	single NN	DMS
RAND	avg	2505.4	2995.4	2845.7	2919.9
	std	104.8	58.2	121.4	51.1
	max	2726	3114	3035	3004
	min	2326	2936	2591	2814
	pred	—	49.8	50.4%	51.4%
ALLD	avg	653.3	999.3	998.3	996.7
	std	133.8	0.5	1.1	2.1
	max	841	1000	1000	1000
	min	507	999	996	997
	pred	—	99.9%	99.9%	99.9%
ALLC	avg	4930.4	4998.6	4996.6	4998.8
	std	209.7	1.0	1.9	1.4
	max	5000	5000	4998	5000
	min	4334	4998	4994	4996
	pred	—	99.9%	99.7%	99.9%
GRIM	avg	724.8	1005.6	1002.0	1001.6
	std	249.5	0.8	2.1	1.8
	max	934	1006	1004	1004
	min	258	1004	999	1000
	pred	—	99.8%	99.5%	99.9%
TFT	avg	1911.7	1005.2	2377.7	2847.3
	std	285.6	1.0	948.8	74.2
	max	2423	1006	2996	2886
	min	1567	1004	1004	2644
	pred	—	99.8%	99.6%	96.8%
STFT	avg	1939.5	1346.3	2181.2	2820.3
	std	358.3	724.8	1017.1	180.9
	max	2487	2722	2994	2886
	min	1285	1000	1000	2306
	pred	—	98.0%	99.8%	97.1%
TFTT	avg	2927.1	1009.0	3639.7	3711.2
	std	783.5	1.0	921.4	44.0
	max	3991	1010	3979	3744
	min	1571	1008	1020	3626
	pred	—	99.7%	98.1%	94.4%
Pavlov	avg	2639.8	2998.4	2992.1	2988.9
	std	448.1	4.7	14.2	8.9
	max	3002	3002	3000	3000
	min	1849	2989	2953	2976
	pred	—	99.4%	99.2%	98.5%

Figure 8. Different methods against CEC'04 IPD default strategies

		D/Y	lin. pred.	single NN	DMS
DMS IPD	avg	2081.3	1001	1000.1	1013.8
		817.3	1013	1019.6	1011.3
	std	575.4	5.8	6.8	7.7
		817.3	6.1	8.4	5.8
	max	2773	1015	1018	1024
		1000	1022	1033	1022
	min	1206	996	994	1002
		593	1000	1008	1006
	pred	85.9%	99.7%	99.6%	99.3%
		—	98.9%	99.2%	99.3%
DMS RPS	avg	—	26.3	86.3	2.7
		—	-26.3	-86.3	-2.7
	std	—	23.7	34.6	26.1
		—	23.7	34.6	26.1
	max	—	55	143	42
		—	14	-42	33
	min	—	-14	42	-33
		—	-55	-143	-42
	pred	—	33.5%	32.1%	28.8%
		—	24.3%	25.2%	28.4%

Figure 9. Results of DMS against different methods (including itself); in each cell first given value belongs to DMS, second to opponent method

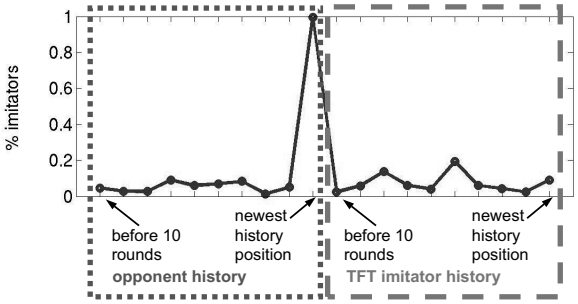


Figure 10. Usage of history positions in PD: DMS against TFT. Left: imitator percentage using positions of opponent history. Right: imitator percentage using positions of own (TFT imitator) history.

All IPD-games except D/Y converged very fast to mutual defection, thus the actions of the players are predicted correctly with very high percentages, but the score is very low. When the DMS played against D/Y, then typically the DMS defected nearly always and D/Y played a very regular pattern, such as "110000110000..." or "10001000...", but sometimes also a more irregular pattern. The average payoff for the DMS was much higher than for D/Y. When the same methods play RPS (with well) against each other, their actions become almost unpredictable and the rate of correct predictions sinks to values in the range of 30%. When playing RPS against itself, the history of the DMS shows no recognizable special pattern and the game ends in about the same payoff for both players. The DMS wins against linear prediction and single net method. Since the D/Y method only works for IPD (because of the strategy coding), we could not compare it with the other methods for RPS.

Concerning the optimization of the history positions, which are used as input for the imitator nets, we observed that good results are also possible when this optimization is switched off and always the 10 previous positions of system and opponent history are used. This result should be expected regarding the relatively simple application task. But the DMS is intended to be applied also to more

complex applications, where a reduction of input complexity is necessary. The optimization of the history positions produces good results, as can be seen in a typical run in Figure 10 for the TFT strategy in the prisoner's dilemma. The history position which contains the last opponent action is used in 98.2% of all considered TFT-imitators, i.e. the 10 best imitators in each optimization step. On average each of these TFT-imitators used 2.0 positions of the opponent history and 1.0 positions of the own history. For simpler strategies like ALLC oder ALLD only 0.8 positions are used on average per history and imitator, whereas more difficult strategies like RAND lead to larger values (3.3 for the opponent history and 4.1 for the own history).

3.1 Prediction Accuracy

At first sight it might be surprising that a better prediction does not always give rise to a better score (see Figure 7, especially for the TFT variants). A similar phenomenon occurs in Figure 9 where the lower correct prediction rate of the DMS against D/Y led to a higher payoff than the higher correct prediction rate of the DMS against the other methods. Should it thus be desirable not to maximize the prediction accuracy in order to achieve the maximal payoff?

Four simplified cases can occur in IPD:

1. opponent difficult to predict; allows low maximal score[1]
2. opponent difficult to predict; allows high maximal score
3. opponent easy to predict; allows low maximal score
4. opponent easy to predict; allows high maximal score

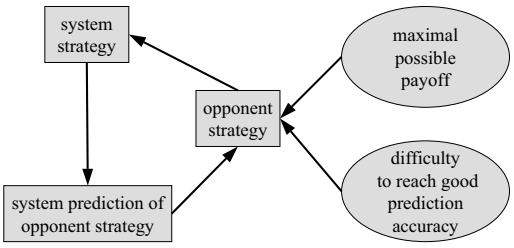


Figure 11. Dependencies in IPD. Arrows: "depends on".

The linear predictor method and TFT only defected against each other. Since a defecting opponent is easy to predict but only allows a low maximal reachable payoff, this belongs to case 3. Accordingly Figure 8 shows a very low average score of 1005.2 and a very high prediction accuracy of 99.8%. The DMS played mostly "C" against TFT in a sophisticated pattern with defective moves in between every now and then. Thus TFT also mostly cooperates in this sophisticated pattern. This is an example of case 2, because the pattern which TFT adopted is difficult to predict, but allows a high maximal score, because it plays mostly cooperation. Here the average payoff is rather high (2847.3), but the prediction accuracy of 96.8% is lower than in the former case.

Since tasks which are difficult to predict mostly come along with a lower prediction accuracy by the predictor, a direct comparison of the cases 2 and 3 might lead to the counterintuitive result that a better prediction accuracy does not result in a better payoff. The problem is, that we cannot compare the results of different runs so easily, because the environment is not stationary. The maximal possible payoff values for IPD (at least for reacting strategies) depend strongly on the opponent strategy, which in turn depends on the own strategy which depends on the own prediction of the opponent strategy. Figure 11 illustrates these cyclic dependencies. And since Figure 7 shows different prediction methods, the results cannot directly be compared. This is true even if the "same" opponent strategy is used. Looking at each situation apart from the others however, an improved prediction accuracy still leads to a higher score. This problem does not occur in RPS where the maximal possible payoff is always 1 point per round – regardless of the opponent strategy. In order to achieve a direct comparability of the results in IPD, it would be desirable to have a normalized prediction accuracy measure, which considers the aforementioned dependencies.

[1]Note: The maximal reachable payoff for a player in IPD depends on the opponent strategy. If the opponent defects, the obtainable score is lower than if the opponent cooperates (cf. Figure 1).

4 Conclusion

The DMS outlined in this paper beats the single neural net in RPS and plays equally good in IPD (mutual defection). Moreover the average payoff against the default strategies in IPD is higher. Since both methods use online learning, one can conclude, that the usage of a *pool* of neural net strategies and the GA optimization of the history windows result in the observed advantage.

The DMS beats the D/Y-method in IPD and plays better against the CEC default strategies. Since both methods use a strategy pool, one can conclude, that our enhancements like online learning, strategy coding with neural nets instead of lookup tables, GA optimization of history windows and explicitly trying out all possible counter-strategies must have brought the advantage. It is planned to analyze in more detail which relative influence is due to which modification.

Despite online-learning, the DMS runs faster than our implementation of the D/Y method (including D/Y offline pool generation). Especially the disassortative sampling in the D/Y method is very time-consuming. Since Darwen and Yao did not measure the diversity or show in [3] how much better their system performs using fitness-sharing and disassortative sampling than without these methods, we did not employ them for the DMS.

As intended, our DMS learns online the current strategy of its opponent and adapts its own strategy accordingly. The performance is very good compared to other playing methods, as has been shown in section 3. Since the DMS is not primarily intended to play games, we want to apply it to other decision making problems, in order to fully exploit its potential.

Acknowledgment

This work was supported by the German Ministry of Education and Research (BMBF) under grant LOKI 01IB001E. We would like to thank Edgar Körner for his support and Paul Darwen for answering several questions concerning the parameters used in his paper [3].

References

[1] M. V. Butz et al., editors. *Anticipatory behavior in adaptive learning systems*. Springer, New York, 2003.
[2] D. Carmel et al. Learning models of intelligent agents. In *Proc. 13th Nat. Conf. AI*, pages 62–67, 1996.
[3] P. J. Darwen and X. Yao. Speciation as automatic categorical modularization. *IEEE Trans. Evolutionary Computation*, 1(2):101–108, 1997.
[4] H. Gross et al. Neural architecture for sensorimotor anticipation. In *EMCSR'98*, pages 593–598, 1998.
[5] T. Ikegami et al. Chaotic itinerancy in coupled dynamical recognizers. *Chaos*, 13(3):1133–1147, 2003.
[6] G. Kendall et al. IPD competition, 2004. Organized at CEC'04, http://www.prisoners-dilemma.com.
[7] C. Meyer et al. Learning strategies in games by anticipation. In *Proc. IJCAI-97*, pages 698–703, 1997.
[8] J. Payne et al. *The adaptive decision maker*. Cambridge University Press, 1993.
[9] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proc. IEEE Int. Conf. Neural Networks*, volume I, pages 586–591. IEEE/INNS, 1993.
[10] W. Schultz et al. A neural substrate of prediction and reward. *Science*, 275:1593–1599, 1997.
[11] D. Suryadi et al. Learning models of other agents using influence diagrams. In *Proc. Int. Conf. User Modeling*, pages 223–232, 1999.
[12] M. Taiji et al. Dynamics of internal models in game players. *Physica D*, 134(2):253–266, 1999.
[13] J. Vidal et al. Recursive agent modeling using limited rationality. In *ICMAS'95*, pages 376– 383, 1995.