Opposition-Based Learning: A New Scheme for Machine Intelligence

Hamid R. Tizhoosh
Pattern Analysis and Machine Intelligence Lab
Systems Design Engineering, University of Waterloo,
Waterloo, Ontario, Canada
Internet: http://pami.uwaterloo.ca/tizhoosh/, E-Mail: tizhoosh@uwaterloo.ca

Abstract

Opposition-based learning as a new scheme for machine intelligence is introduced. Estimates and counter-estimates, weights and opposite weights, and actions versus counter-actions are the foundation of this new approach. Examples are provided. Possibilities for extensions of existing learning algorithms are discussed. Preliminary results are provided.

1. Introduction

Many machine intelligence algorithms are inspired by different natural systems. Genetic algorithms, neural nets, reinforcement agents, and ant colonies are, to mention some examples, well established methodologies motivated by evolution, human nervous system, psychology and animal intelligence, respectively. The learning in natural contexts such as these is generally sluggish. Genetic changes, for instance, take generations to introduce a new direction in the biological development. Behavior adjustment based on evaluative feedback, such as reward and punishment, needs prolonged learning time as well.

Social revolutions are, compared to progress rate of natural systems, extremely fast changes in human society. They occur to establish, simply expressed, the *opposite* circumstances. Revolutions are defined as "..a sudden, radical, or complete change...a fundamental change in political organization;...a fundamental change in the way of thinking about or visualizing something" [1]. Regardless in a scientific, economical, cultural or political sense, revolutions are based on *sudden* and *radical* changes. Of course nobody can guarantee that redirecting a society, a system, or the solution of a complex problem in the opposite direction will necessarily result in a more desirable situation. This is, however, a general phenomenon in machine learning (for instance, mutation can generate lesser fit offspring causing either sub-optimal solutions and/or slower convergence).

In machine intelligence research, there exist numerous paradigms for solving hyper-dimensional problems. Search and optimization techniques (e.g. genetic algorithms), connectionist approaches (e.g. neural nets) and feedback-oriented algorithms (e.g. reinforcement agents) are among mostly used intelligent methods to cope with challenging problems. In this paper, opposition-based learning, as a new learning scheme, will be introduced. In section 2, the basic idea will be described. To illustrate the usefulness of opposition-based learning, in sections 3-5 extensions of three existing paradigms, namely genetic algorithms, reinforcement learning and neural nets, will be briefly introduced and preliminary results will be provided. Section 6 concludes the paper.

2. Basic Idea

Learning, optimization and search are fundamental tasks in the machine intelligence research. Algorithms learn from past data or instructions, optimize estimated solutions and search in large spaces for an existing solution. The problems are from different nature, and algorithms are inspired by diverse biological, behavioral and natural phenomena.

Whenever we are looking for the solution x of a given problem, we usually make an estimate $\hat{x}$. This estimate is not the exact solution and could be based on experience or a totally random guess. The later is usually the case in context of complex problems (e.g. random initialization of weights in a neural net). In some cases we are satisfied with the estimate $\hat{x}$, and sometimes we try further to reduce the difference between the estimated values and the optimal value if the latter is directly or indirectly known. In this sense, if we understand the task of many intelligent techniques as *function approximation*, then we generally have to cope with computational complexity; a solution can be achieved, however, the necessary computation time is usually beyond the permissible application limits (the curse of dimensionality).

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

In many cases the learning begins at a random point. We, so to speak, begin from scratch and move, hopefully, toward an existing solution. The weights of a neural network are initialized randomly, the parameter population in genetic algorithms is configured randomly, and the action policy of reinforcement agents is initially based on randomness, to mention some examples. The random guess, if not far away from the optimal solution, can result in a fast convergence. However, it is natural to state that if we begin with a random guess, which is very far away from the existing solution, let say in worst case it is in the *opposite location*, then the approximation, search or optimization will take considerably more time, or in worst case becomes intractable. Of course, in absence of any a-priori knowledge, it is not possible that we can make the best initial guess. Logically, we should be looking in all directions simultaneously, or more concretely, in the opposite direction. If we are searching for x, and if we agree that searching in opposite direction could be beneficial, then calculating the opposite number $\tilde{x}$ is the first step.

Definition – Let $x \in \Re$ be a real number defined on a certain interval: $x \in [a,b]$. The *opposite number* $\tilde{x}$ is defined as follows:

$$\tilde{x} = a + b - x. \quad (1)$$

For $a = 0$ and $b = 1$ we receive

$$\tilde{x} = 1 - x. \quad (2)$$

Analogously, the opposite number in a multidimensional case can be defined.

Definition – Let $P(x_1, x_2, ..., x_n)$ be a point in a n-dimensional coordinate system with $x_1, ..., x_n \in \Re$ and $x_i \in [a_i, b_i]$. The opposite point $\tilde{P}$ is completely defined by its coordinates $\tilde{x}_1, ..., \tilde{x}_n$ where

$$\tilde{x}_i = a_i + b_i - x_i \quad i = 1, ..., n. \quad (3)$$

The opposition scheme for learning can now be concretized:

Opposition-Based Learning – Let $f(x)$ be the function in focus and $g(\cdot)$ a proper evaluation function. If $x \in [a,b]$ is an initial (random) guess and $\tilde{x}$ is its opposite value, then in every iteration we calculate $f(x)$ and $f(\tilde{x})$. The learning continues with x if $g(f(x)) \geq g(f(\tilde{x}))$, otherwise with $\tilde{x}$.

The evaluation function $g(\cdot)$, as a measure of optimality, compares the suitability of results (e.g. fitness function, reward and punishment, error function etc.).

Considering the unite interval [a1, b1] in Figure 1, the solution for a given problem can be found by repeated examination of guess and counter-guess. The opposite number xo of the initial guess x will be generated. Based on which of the estimate or counter-estimate is closer to the solution, the search interval can be recursively halved until either the estimate or the counter-estimate is close enough to an existing solution.

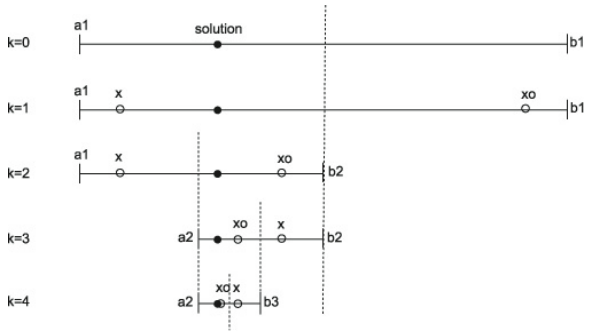

Figure 1. Solving a one-dimensional equation via recursive halving of the search interval with respect to optimality of the estimate x and opposite-estimate xo.

3. Extending Genetic Algorithms

Genetic algorithms (GAs) are stochastic search methods inspired from the natural selection in the evolution process [2,3]. Parameters of an algorithm or system can be regarded as *chromosomes* of individuals in a population of solutions. The fitter solution will be reproduced.

Idea – For every selected chromosome a corresponding *anti-chromosome* can be generated. The initial chromosomes are generally generated randomly meaning that they can possess high or low fitness. However, in a complex problem it is usually very likely that the initial populations are not the optimal ones. In lack of any knowledge about the optimal solution, hence, it is reasonable to look at anti-chromosomes simultaneously. Considering the search direction and its opposite at the same time will bear more likelihood to reach the best population in a shorter time. Specially at the beginning of the optimization, either the chromosome or the anti-chromosome may be fitter (in some cases both may be fit solutions!). Considering a population member and its genetic opponent should accelerate finding the fittest member.

Opposition-Based Extension – Beside the *regular* mutation define a *total mutation* resulting in a complete bit inversion of selected chromosome.

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

Experiments – In each iteration four of the best individuals and four of the weakest members are selected. The anti-chromosomes are generated by total mutation of the weakest members.

Experiment 1 – A continuous and unimodal function should be maximized. The following function with a known maximum at $f(x_{opt}) = 20971520$ is selected:

$$f(x) = \sum_{i=1}^{20} x_i^2 . \quad (4)$$

Results – Table 1 shows improvement caused by the anti-chromosomes used by the opposition-based GA (OGA) compared to the conventional GA. This improvement, however, is not present in results after 50 and 200 iterations. The reason for such can be understood from the graphical display of the iteration progress shown in Figure 2.

Table 1. Results for GA and OGA

	GA	OGA
	After 25 iterations	
$f(x)$	11245058	14083021
σ	1198332	709189
	After 50 iterations	
$f(x)$	16525243	16385800
σ	1112704	616371
	After 200 iterations	
$f(x)$	20462917	20148009
σ	396008	249261

The anti-chromosomes result in a dramatic spike in progress in the first several iterations, when the individuals in the population are relatively far from the global optimum. Hence, in the subsequent iterations, the anti-chromosome may be effectless or even negatively affect the evolution toward optimality. The pattern seen in Figure 2 was repeatedly observed.

Experiment 2 – In order to apply and verify the results obtained from the initial investigations, the more complicated Rosenbrock's valley optimization problem, also known as "Banana" function, was used:

$$f(x) = \sum_{i=1}^{n-1} \left(100 \times \left(x_{i+1} - x_i^2 \right)^2 + \left(1 - x_i \right)^2 \right). \quad (5)$$

Convergence to the global minimum is difficult and hence is used to assess the performance of the anti-chromosomes.

The same effect, namely improvement in early iterations, was observed here as well. Although, the benefit was not as outstanding as in the previous case.

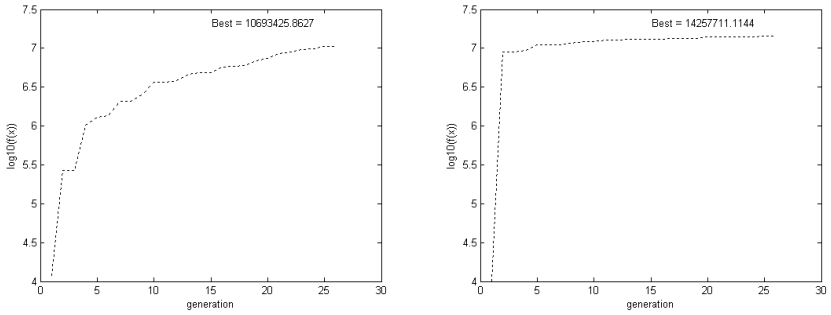

Figure 2. Convergence comparison between GA (top) and OGA (bottom) in a trial.

Other Thoughts – The opposition-based extension of genetic algorithms by means of total mutation seems to be straightforward. However, this can be performed in different ways. In general, if we generate the anti-chromosomes by total mutation, then we are inverting all parameters, from which each chromosome consists. This means that we are making the assumption that all parameters should be changed in order to approach the solution faster. It is possible, however, that only a few parameters have to be changed to their opposite. This brings us to the more logical operation we may call *total sub-mutation*. The total sub-mutation selects parameters within the chromosome and inverts them (Figure 3).

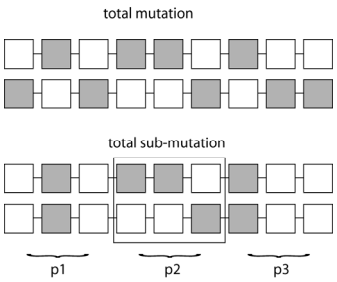

Figure 3. Total mutation versus total sub-mutation. Top: all bits are inverted to mutate to the anti-chromosome, bottom: only a part of the chromosome completely mutates.

4. Extending Reinforcement Learning

Reinforcement learning (RL) is based on interaction of an intelligent agent with the environment by receiving reward and punishment [4]. The agent takes actions to change the state of the environment and receives reward and punishment from the environment. In this sense, reinforcement learning is a type of (weakly) supervised learning. In order to explain how the concept of opposition-based learning can be used to extend reinforcement agents, we focus on the simplest and most popular reinforcement algorithm, namely the Q-learning [4,5].

Generally, the RL agents begin *from scratch* and make stochastic decisions, explore the environment, find rewarding actions and exploit them. Specially at the very begin the performance of the RL agents is poor due to lack of knowledge about which actions can steer the environment in the desired direction.

Idea – Whenever the RL agent takes an action it should also consider the *opposite action* and/or *opposite state*. This will shorten the state-space traversal and should consequently accelerate the convergence.

Opposition-Based Extension – Modification of the Q-learning algorithm involves performing two Q-value updates for each action taken. The conventional algorithm updates the Q-value for the selected action taken at a particular state. The modified algorithm will also update the Q-value corresponding to the action opposite to the one chosen. This "double update" is done in a bid to speed up the learning process.

To accommodate this modification, a *contrariness value* is calculated at each update to determine how different one action is from its nominally opposite action. For instance, in *grid world problem* or *maze learning*, an action that takes the agent to the right by one cell is nominally a direct opposite to the action that takes the agent to the left by one cell. Depending on where the goal is situated, however, the actual reinforcements from the two actions may not be very dissimilar. This situation arises when both actions are equally unfavourable to reaching the goal. In order to take into account the estimated reinforcement of the nominally opposite action without explicitly performing the action, the contrariness value is calculated based on the Q-value of the opposite action to define a corrective measure b for reinforcement modification:

$$b = e^{c}, \tag{6}$$

where c is the contrariness :

$$c = \frac{-2\left|Q(s,a)\right| - Q(s,\tilde{a})}{\max\left(\left|Q(s,a)\right|,\left|Q(s,\tilde{a})\right|\right)}, \tag{7}$$

where s is the state, a the action and $\tilde{a}$ the counter or opposite action. A high contrariness value produces a small b, which is multiplied with the reinforcement received by performing the ordinary action to produce an estimate of the reinforcement for the imaginary opposite action. If the two Q-values are alike, c will be zero and b will be 1, thus assigning the same reinforcement to the opposite action.

The modified algorithm follows the conventional Q-learning algorithm with another crucial difference – updates also need to be performed for the opposite action:

$$Q(s,\tilde{a}) \leftarrow Q(s,\tilde{a}) + \alpha\left(\tilde{r} + \gamma \max Q(s',\tilde{a}') - Q(s,\tilde{a})\right), \tag{8}$$

where reward for opposite action $\tilde{r}$ can be given as

$$\tilde{r} = br. \tag{9}$$

Two sets of experiments were conducted: 1) the Q-matrix is reset after each trial and 2) the matrix is propagated throughout all the trials. They will each be compared to the conventional algorithm to determine the effectiveness of the modification.

Experiment – The program simulates a *grid world* of size 100×100 where an agent has to learn a path to a fixed goal (Figure 4). The starting point at the beginning of each learning trial is randomly decided. The agent successively progresses from one state to another, receiving reinforcements along the way. Reinforcements are inversely proportional to the distance between the goal and the agent. The reinforcement received at each step is used to update the Q-matrix. Learning is considered completed when the agent successfully reaches the goal, which is the terminal state.

Results – The most direct and straightforward criterion for evaluating effectiveness of the algorithms is to observe the number of steps and number of trials required to reach the goal. A lower number of trials indicates that the goal is reached with higher speed and fewer failures. A smaller number of steps means the charted paths are more direct, which is also an indication of faster learning.

The results of the different test runs are presented in Table 2 (average μ and standard deviation σ). From Table 2, noticeable improvement in the convergence behavior of the new algorithm is obvious. The opposition-based extension of Q-learning surpasses the conventional Q-learning in every criterion: average number of steps per trial (407 versus 450), average number of trials required for success (5.6 versus 8.3), and average number of steps per run (2259 versus 3754).

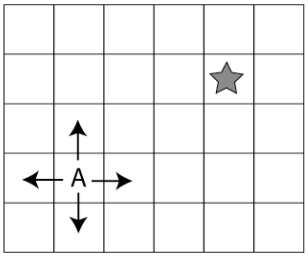

Figure 4. Grid world problem: The agent can move in different directions to reach the goal (cell with star).

Table 2. Results for standard Q-learning (QL) and opposition-based Q-learning (OQL)

	QL			OQL		
	Trials	Average Steps	Total Steps	Trials	Average Steps	Total Steps
μ	**8.25**	**450.10**	**3753.65**	**5.60**	**406.85**	**2259.05**
σ	**6.58**	**258.52**	**3287.61**	**5.09**	**262.12**	**1903.52**

In a third experiment, the two algorithms QL and OQL were run for different grid world sizes. For each grid world two optimal policies π_1 and π_2 were generated manually. A performance measure A was defined to measure the accuracy of the policy π^* generated by the agent:

$$A_{\pi^*} = \frac{\left\| \left(\pi^* \cap \pi_1\right) \cup \left(\pi^* \cap \pi_2\right) \right\|}{m \times m}, \qquad (10)$$

where $\|.\|$ denotes the cardinality of a two dimensional set, and the grid world is of size $m \times m$. For K trials the total average accuracy can be given as

$$\bar{A}_{m \times m} = \frac{1}{K} \sum_{k} A_{\pi^*}^{(k)}. \qquad (11)$$

The results are presented in Table 3.

Table 3. Results of experiments: Average policy accuracies for Q-learning (QL) and the proposed opposition-based algorithm (OQL) for different grid world sizes. A soft-max policy was used for all algorithms. (max. 50 episodes and 1000 iterations per episode).

Algorithms	$\bar{A}_{50\times50}$	σ	$\bar{A}_{100\times100}$	σ
QL	80.5	2.1	66.3	1.0
OQL	**88.1**	2.6	**75.6**	1.4

Other Thoughts – Of course the concept of opposition can be applied if opposite actions and opposite states are meaningful in the context of the problem at hand. The counter-action is defined as "*to make ineffective or restrain or neutralize the usually ill effects of by an opposite force*" [1]. In regard to action a and state s and the existence of their opposites $\tilde{a}$ and $\tilde{s}$, following cases can be distinguished:

- $\tilde{a}$ and $\tilde{s}$ are given: at least four cases can be updated per state observation.
- Only $\tilde{a}$ can be defined: two cases can be updated per state observation.
- Only $\tilde{s}$ can be defined: two cases can be updated per state observation.
- Neither $\tilde{a}$ nor $\tilde{s}$ can be given: application of opposition concept not straightforward.

Assuming that opposite actions and opposite states both exist, then at least four state-action pairs can be updated in each iteration. In general, if action a is rewarded for the state s, then a is punished for the opposite state $\tilde{s}$, the counteraction $\tilde{a}$ is punished for s and rewarded for $\tilde{s}$ (Figure 5).

5. Extending Neural Networks

Artificial neural networks (ANNs) have been established as a major paradigm in machine intelligence research [6,7]. Their ability to learn from samples or instructions, supervised or unsupervised, and from

binary or discrete inputs has made them an unbearable tool in dealing with complex problems.

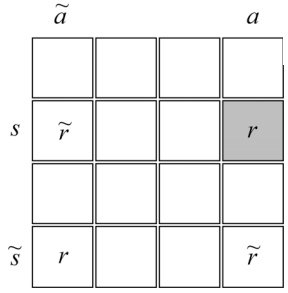

Figure 5. Time saving in RL: the action *a* is rewarded for the state *s* (the gray cell). The opposite cases are updated simultaneously without explicit action-taking.

The idea of opposition-based learning can be integrated into the neural computing in different ways. Generally, we have two possibilities to employ the opposition concept:

- ***Opposite Weights*** – The weights of an ANN can be selected and replaced by opposite weights. This procedure is similar to mutation in genetic algorithms. How many weights should be selected, and how they should be selected offers a wide range of possible schemes to be investigated.
- ***Opposite Nets*** – An instance of the network with opposite weights can be generated. The entire network is duplicated, and all its weights are replaced with opposite weights. In this way we have to observe the error for the net and opposite net and make the decision, which one should be deleted.

5.1 Weights and Opposite Weights

Assuming w_{ij} is the i-th weight of the j-th layer, the randomly selected weight w_{ij} is then changed to its opposite value $\tilde{w}_{ij}$ by

$$\tilde{w}_{ij} = a + b - w_{ij}, \tag{12}$$

where a and b are the minimum and maximum weight values, respectively. Since we may not know the range of every single weight, we have to estimate it. Assuming that $\overline{w}_{ij}(k)$ is the average value for every weight over k consecutive iterations, the interval boundaries can be calculated as follows:

$$a(k) = \overline{w}_{ij}(k)\left(1 - A \times e^{-k}\right), \tag{13}$$

$$b(k) = \overline{w}_{ij}(k)\left(1 + A \times e^{-k}\right), \tag{14}$$

where $A \in \Re^+$ is a sufficiently large number. Hence, at the beginning a large interval is assumed which shrinks as learning proceeds.

5.2 Opposite Net

The other alternative is to create an *opposite network*. An identical instance of the network will be created and all its weights will be replaced by opposite values. The convergence criterion, e.g. the total error of the net and opposite net, will be calculated and compared. The learning will be continued with net or opposite net based on which one has a lower error as described in following algorithm:

Algorithm

1. Initialize the net N
2. Create the opposite net $\tilde{N}$
3. Calculate the errors of each network.
4. If $Error(N) \geq Error(\tilde{N})$, then replace N with $\tilde{N}$
5. Adjust the weights
6. If converged, then stop, else go to step 2

5.3 Experiments with Opposite Weights

As a learning task, the digits 0 to 9 should be recognized by a neural net. The training set consisted of gray-level images of size 20×20. By minor translation and adding noise, the size of training data was increased to 396 images (Figure 6).

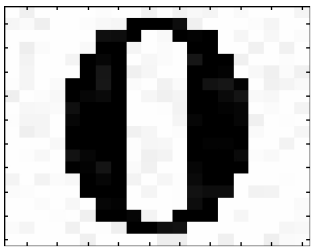

Figure 6. An example from the training set.

In order to demonstrate that the training set is suitable, a conventional network as implemented in Matlab was used. The network had 400 inputs and 2 active layers (20 neurons in the hidden layer and 1 in the output neuron).

For training a resilient backpropagation algorithm was employed. The error function is illustrated in Figure 7, which shows that the selected net dimension was suitable for the digit recognition in our case. We kept this dimension for all other experiments and used this net as a gold standard.

We investigated two different versions in order to verify the usefulness of opposition-based learning for extension of neural nets:

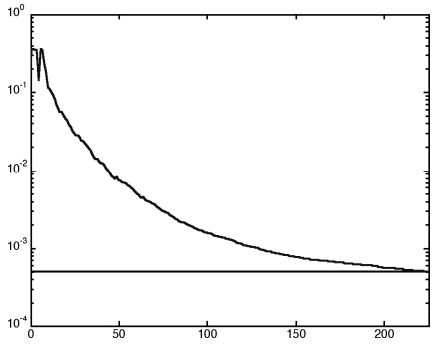

Figure 7. Training with a resilient back-propagation net for digit recognition.

Version 1 – In every learning iteration a opposite net is built consisting of opposite weights. The entire original net is duplicated and its weights are replaced by opposite weights, which are estimated as follows:

$$\tilde{w}_{k,ij} = \frac{2}{l} \times \sum_{n=k-l+1} \left(k w_{n,ij} \right) \times w_{k,ij}, \qquad (15)$$

where k is the learning step, i the neuron, j the input of neuron, and l the length of stored weights (l=3 learning steps).

Version 2 – The second version calculates the opposite weights as follows:

$$\tilde{w}_{k,ij} = \tilde{w}_{k,ij} - 2l \left(\frac{\tilde{w}_{k,ij}}{l} - \sum_{n=k-l+1}^{k} w_{n,ij} \right) \times \frac{a}{e^{k/b} + 1}, \qquad (16)$$

where a and b are selected in such a way that the last term at the beginning almost 10 and after the 50-th learning step equal 1 is. The training was performed 50 times. The learning was interrupted after 200 epochs and the net with lowest error was declared as the winner. In Table 4, the frequency of winning for each net is presented for three different learning rates. Interestingly, in the case of a low learning rate, both versions of revolutionary extension of the network were the best (had the lowest error).

Table 4. The number of times a network had the lowest error

Net	$\alpha = 0.1$	$\alpha = 0.5$	$\alpha = 1.0$
Resilient	0	28	49
Version 1	50	**14**	1
Version 2	50	8	0

6. Conclusions

The concept of opposition-based learning was introduced in this work. The main idea to consider counter-estimates, opposite numbers, anti-chromosomes, counter-actions and opposite weights in machine learning algorithms was discussed. Preliminary results were provided. Considering the magnitude of methodologies and the vast number of possibilities of their modification via opposition concept, the results presented in this work are certainly far from being sufficient or absolute assuring. The preliminary results in this work should solely demonstrate the usefulness of the opposition-based extension for existing algorithms in some cases. Hence, this work does not advocate a final acceptance of opposition-based learning but attempts to demonstrate obvious benefits in a limited manner. Based on observations made, however, one could conclude that the benefits of *revolutionary jumps* during learning has clear advantages in the early stages and will turn into disadvantage as the learning continues. Apparently sudden switching to opposite values should only be utilized at the start to save time, and should not be maintained as the estimate is already in vicinity of an existing solution. Extensive research is still required to implement different concepts, conduct experiments and verify for which tasks and in which situations the proposed learning scheme can bring clear improvement in terms of convergence speedup.

Acknowledgements – The author would like to gratefully thank students for conducting the experiments. Special thanks go to Kenneth Lam, Leslie Yen, Graham Taylor, Monish Ghandi, Rene Rebmann and Jendrik Schmidt.

7. References

[1] Webster Dictionary, www.webster.com.
[2] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, 1989.
[3] M.Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1998.
[4] R.S. Sutton, A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 2003.
[5] C. Watkins, Learning from Delayed Rewards, Thesis, University of Cambidge, England, 1989.
[6] L.V. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms And Applications*, Prentice Hall, 1993.
[7] M. Anthony, *Neural Network Learning: Theoretical Foundations*, Cambridge University Press, 1999.