Neuroevolution of augmented topologies with difference-based mutation

V Stanovov, Sh Akhmedova and E Semenkin
Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russian Federation
E-mail: vladimirstanovov@yandex.ru

Abstract. This study proposes the modification of the neuroevolution of augmented topologies, namely the difference-based mutation operator. The difference-based mutation changes the weights of the neural network by combining the weights of several other networks at the position of the connections having same innovation numbers. The implemented neuroevolution algorithm allows backward connections and loops in the topology, and uses several mutation operators, including connections deletion. The algorithm is tested on a set of classification problems and a rotary inverted pendulum problem and compared to the same approach without difference-based mutation. The experimental results show that the proposed weight tuning scheme allows significant improvements of classification quality in several cases and finding better control algorithms.

1. Introduction
Nowadays the artificial neural networks (NN) are one of the most widely used machine learning tools, which have found many real-world applications in different spheres [1]. The development of feed-forward, convolutional, recurrent and other architectures allowed the basic concept of neural networks to cover a variety of areas where classification, approximation or prediction problems need to be solved [2]. However, despite the advantages, most of the modern architectures are designed by hand by human experts, and creating new efficient architectures could be a challenging task [3].

There are several known approaches to design an efficient architecture of the neural network, and application of evolutionary algorithms (EAs) is one of the most popular approaches [4, 5, 6]. Early studies on evolving neural network architectures have proposed encoding the structure into genetic algorithm (GA) chromosome [7], but more recent investigations have proposed specific encoding schemes. One of the well-known approaches is the NeuroEvolution of Augmented Topologies (NEAT) [8].

In this study the new mutation operator for NEAT algorithm is proposed, called the difference-based mutation. This operator allows improving the search for optimal weight parameters in the architecture of the automatically designed neural network by combining several solutions together. The NEAT algorithm is implemented to solve classification and control problems and tested on several well-known datasets and the rotary inverted pendulum problem.

The rest of the paper is organized as follows: section two describes the NEAT algorithm used in this study, as well as the specific encoding and mutation operators. Section three contains the description of
the proposed modification, section four contains the experimental setup and results, and section five concludes the paper.

2. NeuroEvolution of Augmented Topologies

2.1. Neuroevolution

One of the most important problems in neuroevolution is the solution representation. An individual of the evolutionary algorithm should encode the neurons, their connection structure and weights. Early neuroevolution studies have proposed the Topology and Weight Evolving Artificial Neural Networks (TWEANNs) [9, 10, 11]. Although the TWEANNs allow designing neural architectures, they have a number of disadvantages, such as competing conventions, as highlighted in [1]. The problem of competing conventions means that there are several ways to express a solution into a weight optimization problem of a neural network. The NEAT [8] approach uses a specific encoding and historical memory mechanism, which allows it to solve the competing conventions problem.

Since the development of NEAT many applications of this approach have been found, including control algorithms optimization, classification, regression, playing games, and many more, including evolution of neural architectures [12, 13]. Some recent studies have focused on the landscape analysis of the neuroevolution [14], applying autocorrelation and entropic measure of ruggedness to estimate problem difficulty. In [15] the hyperheuristic approach is proposed to train a neural network that decides which low-level operators to use during the optimization process. In [16] the authors have proposed to use neuroevolution to create a new loss function used in neural networks training process.

The area of NEAT applications is wide, thus the development of novel approaches which would allow improving the performance of the basic algorithm is quite important. In the next subsections first the NEAT encoding is considered, and then the difference-based mutation is proposed.

2.2. NEAT encoding scheme

The NEAT algorithm starts with an initial population of simple networks which then grow by adding new connections or nodes. This approach is different from some others, where the initial population is randomly generated with nodes and connections. In NEAT at the first generation all the inputs are connected to the outputs, and all individuals are equal. The encoding scheme of NEAT during one of the later stages of the search, for a problem with 2 inputs and 1 output, is presented in figure 1.

In figure 1 the hidden node with innovation number 57 is added during the search process, and it is 57-th mutation applied since the beginning of the search. All nodes, except for the hidden one have no operation \((x = x)\), while the hidden node performs squaring operation: all inputs at every step are summed and squared. The connection genes also hold the innovation numbers, as well as the source (from which node), destination (to which node), weight coefficient, and activation flag, i.e. whether this connection is active or not. Three connections were added during the search and have innovation numbers 84, 107 and 152. Note that connection 107 is connected to itself, i.e. it creates a loop, and all values at the output of node 57 are sent to the input. The graphical representation of this individual is presented below, showing the nodes and connections as a directed multigraph.

The encoding scheme presented in [8] allows starting with a minimal network and thus begin the search in smaller dimensional space, unlike the TWEANNs. The number of nodes and connections has no strict limitations, which makes NEAT an evolutionary algorithm with varying chromosome length, and requires specific mutation and crossover operations to be used. The calculations of the output values are repeated during several steps until all reachable nodes are activated and the signal from them is propagated to the output nodes.
### NEAT mutation and crossover operators

The original study on NEAT proposed two main mutation operations: adding connections and adding nodes to connections. In this study the following operators were used:

- Adding connection. Two random nodes are selected and the connection is created between them, with a random weight assigned from \( \text{randnorm}(0,0.1) \) or \( \text{randnorm}(1,0.1) \) with 0.5 probability. The destination node cannot be the bias or input node. Probability to use: 0.1.
- Mutating random node. The operation type of a randomly selected node is changed to another one. Probability to use: 0.05.
- Removing connection. If there are at least several connections in the solution, one of them is randomly selected and removed. Probability to use: 0.15.
- Merging connections. If there are two nodes linked to each other by to connections, these connections are merged into one, with the weight vector equal to the sum of their weights. Probability to use: 0.4.
- Adding node to connection. A random connection is selected and divided into two, with a node between them, and a random operation type is assigned to this node. One of the weight either to the new node or from it is set to 1, while the other is set to be the same as previous. Probability to use: 0.1.
- Random weights. With probability equal to \( 1/\text{NumberOfConnections} \) the random weight is assigned to one or several connections, generated from either \( \text{randnorm}(0,0.1) \) or \( \text{randnorm}(1,0.1) \) with 0.5 probability. Probability to use: 0.1.

**Figure 1.** Example of solution encoded in NEAT.

The mutation and crossover operators of NEAT used in this study are presented in the next subsection.

| Individual | Node 0 | Node 1 | Node 2 | Node 3 | Node 57 |
|------------|--------|--------|--------|--------|---------|
| Bias       | x=x    | Input  | Input  | Output | Hidden  |
|            |        | x=x    | x=x    | x=x    | x=x     |
| Conn 0     | From 0 | To 3   | Weight 0.5 | Active 1 |
| Conn 1     | From 1 | To 3   | Weight 0.1 | Active 1 |
| Conn 2     | From 2 | To 3   | Weight 0.7 | Active 1 |
| Conn 84    | From 1 | To 57  | Weight 0.2 | Active 1 |
| Conn 107   | From 57| To 3   | Weight 0.1 | Active 1 |
| Conn 152   | From 57| To 57  | Weight 1.0 | Active 1 |

| Nodes       | Weights | Actives |
|-------------|---------|---------|
| Node 0 Bias | 0.500   | 1       |
| Node 1      | 0.100   | 1       |
| Node 2      | 0.100   | 1       |
| Node 3 Output| 0.200   | 1       |
| Node 57 Hidden| 1.000   | 1       |
| Node 152    | 1.000   | 1       |

The diagram shows the connections and weights between the nodes.
After the mutation operation, it is possible that there are invalid nodes or connections – not connected to anything or going to non-existing nodes, these parts of the solution are removed. Each time one of the mutation operators is applied, the newly generated nodes or connections receive a global innovation number $\text{Inn}$, which is incremented after every operation. Some operations may generate more than one innovation. Only one of the 6 mutation operators is used for every individual.

The crossover operator combines the genetic information of two parents to produce new offspring for further mutation. In NEAT the historical markings, e.g. innovation numbers are used to match same genes with each other. When two individuals are crossed, first the mating genes are aligned, i.e. the parts of the chromosomes which are the same for both parents starting from the beginning. Once there is a difference found in the genotypes, the following genes are considered as disjoint or excess, depending on whether they are within or outside the range of the innovation numbers of the other parent. To produce an offspring the genes are randomly selected from either first or second parent, and the disjoint and excess genes are taken from the parent with higher fitness. More detailed description of the crossover operator is given in [8].

### 2.4. NEAT general scheme

The original NEAT algorithm uses the speciation mechanism to protect innovation, however, in this study a different approach is used to manage the population. The algorithm scheme used in this study is presented below:

1. Initialize population $P$ with minimal solutions, calculate fitness $F_i, i = 1, ..., N$

   1.1. For generation number $g = 1, ..., NG$

   1.2. For all individuals $i = 1, ..., N$ in the population:

      1.2.1. Randomly select random indexes $r1$ and $r2$

      1.2.2. Perform crossover with $P_{r1}$ and $P_{r2}$ to produce $OFS$

      1.2.3. Select one of mutation operators used

      1.2.4. Perform mutation on $OFS$

      1.2.5. Calculate fitness of offspring $F(OFS)$

      1.2.6. If $F(OFS) < F_i$ then replace $P_i$ with $OFS$, replace $F_i$ with $F(OFS)$

2. Find and return the best individual.

The algorithm scheme shows that the new solutions are accepted only if they perform better than previous ones.

The implementation used in this study had the following main activation functions, where $x$ is the sum of all inputs:

- Linear: equals $x$
- Unsigned step function: equals 1 if $x > 0$
- Sine: $\sin(PI*x)$
- Gaussian: $\exp(-x*x/2)$
- Hyperbolic tangent: $\tanh(x)$
- Sigmoid: $(\tanh(x/2)+1)/2$
- Inverse: $-x$
- Absolute value: $|x|$
- ReLU: $\max(0, x)$
- Cosine: $\cos(PI*x)$
- Squared: $x^2$

The default function is linear (equal), in case if mutation operation generated a new node, one of these activation functions was assigned randomly.
3. Proposed approach

One of the main problems of the NEAT algorithm is the lack of an efficient parameter tuning method. As long as the neural network structure may contain loops and backward connections, the classical tuning methods, such as backpropagation algorithm, could not be used directly. Another option is to use the direct search methods, such as genetic algorithms [12], particle swarm optimization [13], differential evolution [14] or other approaches [15] for weights tuning. The disadvantage of this approach is that it would require tuning each individual separately, which may result in significant computational overhead. Several approaches have been proposed to solve this problem, for example, the cooperative coevolution of synapses [21], which was shown to significantly improve the algorithm capabilities.

In this study the new difference-based mutation scheme is proposed, inspired by the mutation operator in differential evolution algorithm. The visual representation of difference-based mutation is shown in figure 2.

![Solution 1](image1)

![Solution 2](image2)

![Solution 3](image3)

![Solution 4](image4)

**Figure 2.** Identifying same genes to perform the difference-based mutation.

The difference based mutation for a given individual $P_i$ selects three randomly chosen individuals in the population with indexes $r1$, $r2$ and $r3$ and searches for the connections genes with equal innovation.
numbers. The positions of these numbers are stored in vectors $e_i$, $e_{r1}$, $e_{r2}$ and $e_{r3}$, as long as the same innovation numbers could be located in different parts of the chromosome. Random indexes are selected so that $I \neq r1 \neq r2 \neq r3$. The genes having the same weight values are not considered. If there is at least one innovation number for all 4 individuals, the following procedure is used to generate new weights for the offspring individual $OFS$:

$$w_{e_{OFS}} = w_{e_{r1}} + F \cdot (w_{e_{r2}} - w_{e_{r3}})$$

where $F$ is the scaling factor parameter, and $w_e$ are weight vectors of corresponding individuals. The main idea of difference-based mutation is to use the information about weights from other individuals to allow efficient steps in the parameter search space. This mutation operator only changes the shared weight parameters, which means that these parameters are probably important for the performance of the final solution, as long as they are represented in several of them. The difference-based mutation is used alongside with the other mutation operators described in the previous section with probability of 0.1. In figure 2 four solutions participate in the mutation process. In each solution there is a connection node with innovation number equal to 84, which means that it is possible to perform difference-based mutation. All genes with innovation number 84 have difference weight, which could be achieved by applying random weights mutation or difference-based mutation. Other genes do not participate in the mutation, as long as they either have different innovation numbers, e.g. 107, 53, 182, 117, etc., or have equal weights. The next section contains the experimental setup, algorithm parameters and results.

4. Experimental setup and results
Two sets of experiments have been performed to estimate the efficiency of the proposed approach. First, the classification problems have been used; in particular, the set of databases from the KEEL repository was used, with the characteristics shown in table 1.

| Dataset      | Number of instances | Number of features | Number of classes |
|--------------|---------------------|--------------------|-------------------|
| Australian credit | 690                 | 14                 | 2                 |
| German credit       | 1000                | 24                 | 2                 |
| Segment            | 2310                | 19                 | 7                 |
| Phoneme            | 5404                | 5                  | 2                 |
| Page-blocks        | 5472                | 10                 | 5                 |
| Twonorm            | 7400                | 20                 | 2                 |
| Ring               | 7400                | 20                 | 2                 |
| Magic              | 19020               | 10                 | 2                 |

The datasets in table 1 have different characteristics and are taken from different fields, some of them have large number of classes, others are characterized by class imbalance, large number of instances, which make these problems difficult to solve for most methods. Table 2 presents the averaged training accuracy results for the all used datasets.

| Dataset      | NEAT  | NEAT-DBM |
|--------------|-------|----------|
| Australian credit | 0.864 | 0.865    |
| German credit       | 0.745 | 0.757    |
| Segment            | 0.484 | 0.613    |
| Phoneme            | 0.765 | 0.771    |
| Page-blocks        | 0.917 | 0.926    |
| Twonorm            | 0.897 | 0.926    |
| Ring               | 0.701 | 0.719    |
| Magic              | 0.789 | 0.797    |
The categorical cross-entropy was used as a criteria to be optimized by both standard NEAT and NEAT with difference-based mutation (NEAT-DBM). The population size was set to 100 individuals, and the algorithm ran for 500 generations. To limit the size of the solutions the upper limit on the number of connections was set to 350. For every dataset one run of the 10-fold cross-validation was performed, and the accuracy values were calculated both NEAT and NEAD-DBM. The results in table 2 show that the NEAT-DBM allowed improvements of training accuracies on all datasets used in the experiments. For some datasets, like Australian credit, the improvement is relatively small, while for some others, like Segment dataset, the difference is quite big. Table 3 contains the averaged test accuracy of the original and modified approaches.

Table 3. Accuracy of NEAT and NEAT-DBM, test set.

| Dataset       | NEAT  | NEAT-DBM |
|---------------|-------|----------|
| Australian credit | 0.859 | 0.866    |
| German credit | **0.744** | 0.738    |
| Segment       | 0.477 | **0.622** |
| Phoneme       | 0.759 | **0.768** |
| Page-blocks   | 0.918 | **0.924** |
| Twonorm       | 0.893 | **0.924** |
| Ring          | 0.713 | **0.715** |
| Magic         | 0.791 | **0.795** |

As the results in table 3 show, the advantage of NEAT-DBM on the test set is smaller than on the training set, and for the German dataset the accuracy is even smaller.

Table 4 shows the results of the Mann-Whitney statistical tests comparing the efficiency of the NEAT and NEAT-DBM on both training and test sets. The values in the table 4 are the test result and the standard score $Z$ value (0 = equal, 1 = better), NEAT is taken as baseline. The values of $Z$ larger the 1.96, were considered as significant improvements, i.e. significance level $p = 0.05$.

Table 4. Statistical comparison of NEAT and NEAT-DBM.

| Dataset       | NEAT  | NEAT-DBM |
|---------------|-------|----------|
| Australian credit | 0, 0.037 | 0, 0.68 |
| German credit | 0, 1.93 | 0, -0.34 |
| Segment       | **1, 3.40** | **1, 3.59** |
| Phoneme       | 0, 0.75 | 0, 0.87 |
| Page-blocks   | **1, 2.94** | 0, 1.02 |
| Twonorm       | 0, 1.88 | **1, 2.38** |
| Ring          | 0, 1.28 | 0, 0.15 |
| Magic         | 0, 1.88 | 0, 0.75 |

Considering the results in table 4, it could be concluded that the difference-based mutation operator allowed significant improvements of the performance of the NEAT algorithm on Segment dataset, both training and test sets, as well as Page-blocks dataset and Twonorm dataset.

Figure 3 shows an example of neural structure generated by the algorithm for the Phoneme dataset.
Figure 3. One of the solutions to the Phoneme dataset, test accuracy is 0.76

Figure 3 shows that NEAT is capable of designing relatively compact models with few connections (requires only 26 operations to calculate the output), which allows solving the classification problems with minimum computational requirements.

The second set of experiments was performed on the problem of stabilizing the rotary inverted pendulum (RIP). The RIP controller design [22] is often considered as a test problem in the control community, due to its properties, which include nonlinearity, non-minimum phase and instability, which make it highly difficult to control. It could be considered as attitude control of a space booster rocket, automatic aircraft landing system, stabilization of a cabin in a ship, etc. [23]. Therefore, an algorithm capable of controlling this system could be adapted to more complex real-world systems.

There are several known typical control problems which include pendulum stabilization. In RIP problem the pendulum is mounted on the output arm, which is driven by a servo motor. The pendulum position and the arm position are measured by encoders. The input to the system is the servo motor direction and voltage. The pendulum has one stable equilibrium point, when the pendulum is pointing down, and another unstable point, where it is pointing up, and the goal is to keep it up and turn the arm to the desired angle. In this study, the model described in [22] is used for experiments, where the state variables of the RIP system are:

\[ [x_1, x_2, x_3, x_4]^T = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T. \]

The input to the controller at every time moment consisted of these four state variables. Figure 4 shows the RIP and its main characteristics.

Figure 4. Rotary inverted pendulum.

\[ \text{Figure 3. One of the solutions to the Phoneme dataset, test accuracy is 0.76} \]
The full list of parameters and their values was taken from [22] and is provided in table 1. The modelling was performed with step size of \( h = 0.01 \) and time \( T_{\text{max}} = 10 \) seconds. During modelling, the number of revolutions of arm and pendulum was also measured.

As long as the goal of the control was to set the pendulum to the upright position and the arm to the predefined zero position, the fitness calculation was based on the total difference between the desired and actual positions of the pendulum and the arm, as well as their angular velocity. So, the fitness was defined as follows:

\[
f_i = \frac{1}{\pi} \sum_{j=0}^{T_{\text{max}}} |x_1^j| + \frac{1}{\pi} \sum_{j=0}^{T_{\text{max}}} |x_2^j| + \frac{1}{2} \sum_{j=0}^{T_{\text{max}}} |x_3^j| + \frac{1}{20} \sum_{j=0}^{T_{\text{max}}} |x_4^j|
\]

Each state variable in fitness calculation was weighted by its range. The controller design consisted of four separate steps, where the goal of the first step was to stabilize the pendulum in upright position, the second was to stabilize and bring arm to the desired position, and the third and fourth were to repeat the same with swing-up sequence. The parameters of the starting points for the control are presented in table 6.

**Table 5. Rotary inverted pendulum parameters.**

| Name | Description | Value |
|------|-------------|-------|
| \( m_1 \) | Mass of arm | 0.056 kg |
| \( l_1 \) | Length of arm | 0.16 m |
| \( c_1 \) | Distance to arm center of mass | 0.08 m |
| \( J_1 \) | Inertia of arm | 0.00215058 kg.m^2 |
| \( m_2 \) | Mass of pendulum | 0.022 kg |
| \( l_2 \) | Length of pendulum | 0.16 m |
| \( c_2 \) | Distance to pendulum center of mass | 0.08 m |
| \( J_2 \) | Inertia of pendulum | 0.00018773 kg.m^2 |
| \( R_m \) | Armature resistance | 2.5604 \( \Omega \) |
| \( K_b \) | Back-emf constant | 0.01826 V.s/rad |
| \( K_t \) | Torque constant | 0.01826 N.m/A |
| \( \theta_1 \) | Angular displacement of arm | - |
| \( \theta'_1 \) | Angular velocity of arm | - |
| \( \theta_2 \) | Angular displacement of pendulum | - |
| \( \theta'_2 \) | Angular velocity of pendulum | - |
| \( \tau \) | Applied torque | - |

**Table 6. Starting positions for experiments.**

| Starting position | \( x_1 \) | \( x_2 \) | \( x_3 \) | \( x_4 \) |
|------------------|----------|----------|----------|----------|
| 1                | 0        | -0.1     | 0        | 0        |
| 2                | 3        | -0.1     | 0        | 0        |
| 3                | 0        | 1.41     | 0        | 0        |
| 4                | 3        | 1.41     | 0        | 0        |

The learning process was divided in four parts, the maximum number of generations was set to 1000, and the population size was set to 200 individuals. To test the efficiency of the approach there were 40 independent runs of the algorithm performed. For every run, at each generation the best individual was used to build the graphs of control. Figures 5 and 7 show the best obtained controllers behavior by neuroevolution with and without difference-based mutation, and figures 6 and 8 show the architecture of the trained nets.
Figure 5. Pendulum dynamics with controller designed by neuroevolution without DBM.

Figure 6. Best controller designed by neuroevolution without DBM.
Figure 7. Pendulum dynamics with controller designed by neuroevolution with DBM.

Figure 8. Best controller designed by neuroevolution without DBM.

Figure 5 shows that the controller designed without DBM was able to stabilize the pendulum itself in all four scenarios, however, it was not able to set the arm in the desired zero position or at least minimize its speed. However, the controller designed with DBM (figure 7) was able to stabilize both the arm and the pendulum, although it could not move the arm to the desired zero position.

The architecture of the network, shown in figure 6 in this case, contains a lot of connections and is relatively complex (13 nodes, 27 connections), compared to the architecture of the network, designed with distance-based mutation (11 nodes, 20 connections).

The averaged fitness value over 40 independent runs for the controllers designed by neuroevolution without DBM was equal to $394.26 \pm 73.90$, while the average fitness of controllers designed with DBM was equal to $378 \pm 81.61$, i.e. the modification allowed creating more efficient controllers on average.
5. Conclusions
In this paper a new mutation mechanism was proposed for the neuroevolution of augmented topologies. The difference-based mutation uses the information about the values of the weight vectors of connections of different individuals in the population to allow directed search in the parameter space. The proposed approach was applied to the NEAT solving several classification problems and a control problem, however, the difference-based mutation is a general operator, and could be used for other realizations of NEAT framework. The experimental results have shown the efficiency of the proposed modification, it allowed significant classification accuracy improvements with the same computational resource in terms of number fitness calculations on both training and test sets in two cases out of nine. Also, the application of distance-based mutation has allowed designing more efficient controllers for the rotary inverted pendulum, in particular, better results were achieved for both best and average cases. Further directions of studies may include using the difference-based mutation in NEAT to other types of problems, which are typically solved by neuroevolution, such as regression or control problems.

Acknowledgments
This work was supported by the Ministry of Science and Higher Education of the Russian Federation (State Contract No. FEFE-2020-0013)

References
[1] Schaffer J D, Whitley D and Eshelman L J 1992 Combinations of genetic algorithms and neural networks: A survey of the state of the art. Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN-92) 1-37
[2] Abbass H A 2003 Pareto neuro-evolution: Constructing ensemble of neural networks using multi-objective optimization Evolutionary Computation, CEC’03 3 2074-80
[3] Chenxi L, Zoph B, Neumann M, Shlens J, Hua W, Li L-J, Fei-Fei L, Yuille A, Huang J and Murphy K 2018 Progressive Neural Architecture Search Proceedings of the European Conference on Computer Vision (ECCV 2018) 19-34
[4] Real E, Moore S, Selle A, Saxena S, Suematsu Y L, Tan J, Le Q V and Kurakin A 2017 Large-scale evolution of image classifiers ICML
[5] Miikkulainen R, Liang J Z, Meyerson E, Rawal A, Fink D, Francon O, Raju B, Shahrzad H, Navruzyan A, Duffy N and Hodjat B 2017 Evolving deep neural networks arXiv: Neural and Evolutionary Computing 293-312
[6] Xie L and Yuille A L 2017 Genetic CNN IEEE International Conference on Computer Vision 1388-97
[7] Angeline P J, Saunders G M and Pollack J B 1993 An evolutionary algorithm that constructs recurrent neural networks IEEE Transactions on Neural Networks 5 54-65
[8] Stanley K O and Miikkulainen R, 2002 Evolving Neural Networks through Augmenting Topologies Evolutionary Computation 10(2) 99-127
[9] Braun H and Weisbrod J 1993 Evolving feedforward neural networks Proceedings of ANNGA93 International Conference on Artificial Neural Networks and Genetic Algorithms 25-32
[10] Dasgupta D and McGregor D 1992 Designing application-specific neural networks using the structured genetic algorithm Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks 87-96
[11] Yao X and Liu Y 1996 Towards designing artificial neural networks by evolution Applied Mathematics and Computation 91(1) 83-90
[12] Floreano D, Durr P and Mattiussi C 2008 Neuroevolution: From Architectures to Learning Evolutionary Intelligence 1 47-62
[13] Lehman J and Miikkulainen R 2013 Neuroevolution Scholarpedia 8(6) 30977
[14] Rodrigues N M, Silva S and Vanneschi L, A Study of Fitness Landscapes for Neuroevolution 2020 IEEE Congress on Evolutionary Computation (CEC) 1-8
In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (GECCO '20) 111-2

[15] Gonzalez S and Miikkulainen R, Improved Training Speed, Accuracy, and Data Utilization Through Loss Function Optimization 2020 IEEE Congress on Evolutionary Computation (CEC) 1-8

[16] Deb K and Deb D 2014 Analysing mutation schemes for real-parameter genetic algorithms International Journal of Artificial Intelligence and Soft Computing 4(1) 1-28

[17] Kennedy J and Eberhart R 1995 Particle Swarm Optimization IEEE International Conference on Neural networks 1942-8

[18] Storn R and Price K 1997 Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces Journal of Global Optimization 11(4) 341-59

[19] Yang X S and Deb S 2009 Cuckoo Search via Levy flights World Congress on Nature & Biologically Inspired Computing 210-4

[20] Gomez F, Schmidhuber J and Miikkulainen R 2008 Accelerated Neural Evolution through Cooperatively Coevolved Synapses Journal of Machine Learning Research 9 937-65

[21] Fairus M A, Mohamed Z and Ahmad M N, Fuzzy modeling and control of rotary inverted pendulum system using LQR technique IOP Conf. Series: Materials Science and Engineering 53 1-11

[22] Prasad L B, Tyagi B and Gupta H O Optimal Control of Nonlinear Inverted Pendulum System Using PID Controller and LQR: Performance Analysis Without and With Disturbance Input International Journal of Automation and Computing 11(6) 661-70