Biuletyn WAT
Vol. LX, Nr 2, 2011

Forming dynamic, self-organizing neural networks by means of Assembler Encoding

TOMASZ PRACZYK

Polish Naval Academy, Institute of Naval Weapon, Śmidowicza 69 Str. , 81-103 Gdynia

Abstract. Assembler Encoding is a new neuro-evolutionary method. To date, it has been tested in such problems as: an optimization problem, a predator-prey problem, and in an inverted pendulum problem. In all the cases mentioned, Assembler Encoding was used to create neural networks with constant, invariable architecture. To test whether Assembler Encoding is able to form other types of neural networks, next experiments were carried out. In the experiments, the task of Assembler Encoding was to form self-organizing, dynamic neural networks. The networks were tested in the predator-prey problem. To compare Assembler Encoding with other method, in the experiments, a modified version of standard neuro-evolution was also applied. The results of the experiments are presented at the end of the paper.
Keywords: self-organizing neural networks, neuro-evolution, predator-prey problem

1. Introduction

Assembler Encoding (AE) is a new neuro-evolutionary method. It originates from the cellular [4] and edge encoding [5], although it also has features common with Linear Genetic Programming presented, among other things, in [7]. AE represents Artificial Neural Network (ANN) in the form of a program (Assembler Encoding Program – AEP) whose structure is similar to the structure of a simple assembler program. The task of AEP is to create Network Definition Matrix (NDM) containing all the information necessary to produce ANN. The process of ANN construction consists of three stages (Fig. 1). First, Genetic Algorithm (GA) produces AEPs. Then, each AEP creates and fills up NDM. Finally, NDM is transformed into ANN.

To test AE, different experiments were carried out [10, 11, 12]. In the experiments, the task of AE was to construct ANNs with constant architecture (static ANNs).

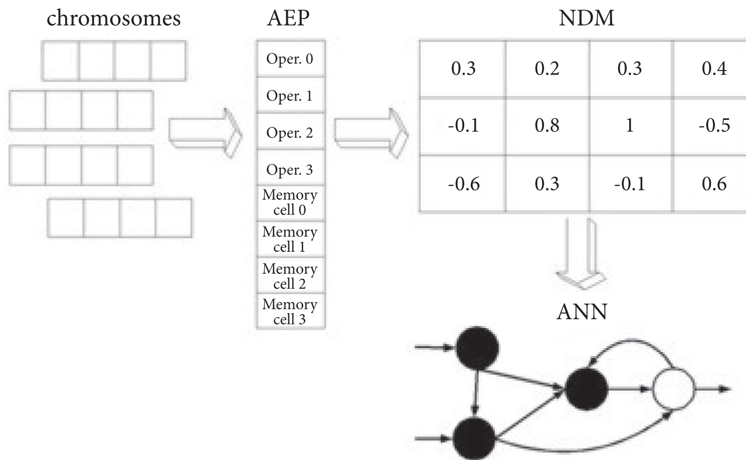


Fig. 1. Creating ANN in AE

Such ANNs do not change over time. Both weights and topology of the static ANNs are invariable during whole their lifetime. All the experiments showed that AE is quite effective method for creating such ANNs. However, the question is whether AE is a universal tool and whether it can also be used to successfully create other types of ANNs. To answer this question, next experiments were carried out. In the experiments mentioned, the so-called dynamic ANNs were used [2, 14]. The dynamic ANNs are self-organizing networks in which weights of interneuron connections change over time. The magnitude of change depends on the type of Hebb rule assigned to a given connection and additionally on the value of a learning rate. The main reason why the dynamic ANNs were selected to tests is their great ability to transfer from simulation to reality [14] as well as an intention to use them in a real system to control autonomous underwater vehicles (AUV).

The experiments with the dynamic ANNs were performed in the predator-prey problem. In the experiments, the goal of ANNs was to control predators responsible for capturing a fast moving prey behaving according to a simple deterministic strategy. Due to the difference in movement speed between the predators and the prey, the former had to cooperate in order to capture the latter. To compare AE with other method, in the experiments, a co-evolutionary version of the standard neuro-evolution (SNE) was also tested. In the paper, SNE is the method in which all the information necessary to construct ANN is stored in a single chromosome and all chromosomes encoding ANNs are grouped in a single population. In the co-evolutionary version of the method above, all ANNs evolve in a few separate populations.

The main motivation to use the predator-prey problem as a test bed for AE is its similarity to the real problem in which the task of a group of AUVs is to guard an entrance to a harbour and to capture all dangerous underwater objects that can

appear near the guarded area. The problem described above is the target problem that is planed to be solved within the confines of the future research.

The paper is organized as follows: section 2 outlines AE; section 3 presents a variant of AE used to create the dynamic ANNs, section 4 illustrates the results of the experiments; section 5 is the summary.

2. Fundamentals of Assembler Encoding

In AE, ANN is represented in the form of AEP, which is composed of two parts, i.e. a part including operations (the code part of AEP) and a part including data (the memory part of AEP). The task of AEP is to create and to fill in NDM with values. To this end, AEP uses predefined operations which are run one after another. When working the operations use data located at the end of AEP. Once the last operation finishes its work, the process of creating NDM is completed. The matrix is then transformed into ANN.

To create ANN based on NDM, the latter has to include all the information necessary to create the network. In the simplest case, i.e. for the static ANNs, NDM takes the form of the classical Connectivity Matrix (CM) [6]. Elements of such a matrix determine synaptic weights between neurons. For example, NDM[*i*, *j*] defines the link from neuron *i* to neuron *j*. Apart from the basic part, NDM can also include additional columns that describe parameters of neurons, e.g. type of neuron (sigmoid, radial), bias etc.

2.1. Operations

AEPs can use various operations. The main task of most operations is to modify NDM. The modification can involve a single element of the matrix or a group of elements. Figures 2 and 3 present the implementation of two example operations.

`CHGC0` presented in Fig. 2 belongs to the class of binary encoded operations (encoding operations is explained in further part of the paper). It modifies elements of NDM located in a column indicated by the parameter p_0 and the register R_2 (in AE two registers are used, i.e., R_1 and R_2; the role of the registers is explained further). An index of the first element updated is located in the register R_1 whereas a total number of elements updated by the operation is stored in the parameter p_2. To modify elements of NDM, `CHGC0` uses data from AEP. An index to a memory cell including the first data used by `CHGC0` is stored in p_1.

`CHG_#2` presented in Fig. 3 belongs to the class of operations encoded in the form of strings including zeros, ones and the so-called *don't cares* denoted as "#" [1, 3]. The operation modifies elements of NDM indicated in `list1` and `list2`.

```
CHGC0(p0,p1,p2,p3)
{
column=(abs(p0)+R2)mod NDM.height;
numberOfIterations=abs(p2)mod NDM.width;
for(i=0;i<=numberOfIterations;i++)
      {
      row=(i+R1)mod NDM.width;
      NDM[row,column]=D[(abs(p1)+i)mod D.length]/Max_value;
      }
}
```

Fig. 2. Pseudo-code for CHGC0 (NDM[i, j] is element of NDM, P_i $i = 1,2$ is a value of the ith register, Max_value is scaling value which scales all elements of NDM to range <–1, 1>, `D[i]` is the ith data, `D.length` is a number of memory cells)

```
CHG_#2(p0,list1,list2)
{
for(i=0;i<list1.length;i++)
      for(j=0;j<list2.length;j++)
            {
            row=(list1[i]+R1)mod NDM.width;
            column=(list2[j]+R2)mod NDM.height;
            NDM[row,column]=
            D[(abs(p0)+i*list2.length+j)mod D.length]/Max_value;
            }
      }
```

Fig. 3. Pseudo-code for `CHG_#2`

The lists mentioned include numbers of columns and rows of NDM (`list1` includes numbers of rows while `list2` contains numbers of columns) which, in turn, indicate elements of the matrix that are updated as a result of execution of the operation. All possible combinations of columns and rows considered in both lists determine a set of elements modified by the operation. p_0 indicates a place in the memory part of AEP where new values for updated elements can be found.

In addition to operations whose task is to modify a content of NDM, AE also uses a jump operation denoted as `JMP`. The jump makes it possible to repeatedly use the same code of AEP in different places of NDM. It is possible thanks to changing values of the registers once the jump is performed. Each jump determines a place in the code part of AEP where processing should continue. It also determines a number of jumps and a place in the memory where new values for the registers are placed. Figure 4 shows the situation in which the jump is run two times. The sequence of two operations (Operation 0 and Operation 1) is executed three times, but each time in a different place of NDM. The first time, the operations are executed for initial values of the registers. The second time, after the first activation of the jump, the registers are changed to $R_1 = 0$, $R_2 = 2$. The last execution of the two operations is connected with the following values of the registers: $R_1 = 2$, $R_2 = 2$.

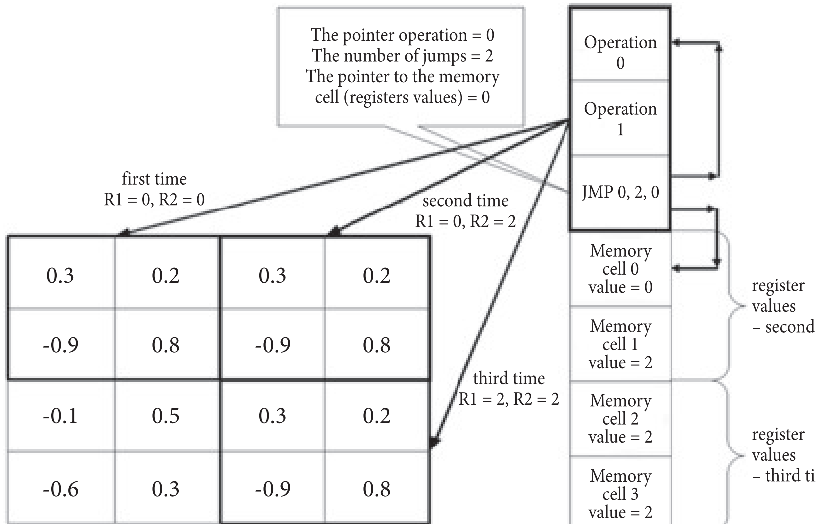


Fig. 4. JMP operation

2.2. Evolution in AE

In AE, AEPs and in consequence ANNs are created by means of GA. The evolution of AEPs proceeds according to the scheme which is an adaptation of the idea of evolving co-adapted subcomponents proposed by Potter and De Jong [8, 9]. To create AEP, the scheme mentioned combines operations and data from various populations. Each population including chromosomes-operations (each chromosome-operation encodes the type of operation, e.g. `CHGC0`, and parameters of operation; implementations of operations do not evolve) has a number assigned determining a position of an operation from the population in AEP. In this approach, the number of operations corresponds to the number of populations including the chromosomes-operations. Each population delegates exactly one representative to each AEP. In the beginning, AEPs have only one operation and a sequence of data. Both the operation and data come from two different populations. Further populations including operations are successively added if generated AEPs cannot accomplish progress in performance over an assumed number of co-evolutionary cycles (the term "co-evolutionary cycle" is used to differ it from the evolutionary generation taking place inside a single population with operations and data). The populations with operations and data can also be replaced by newly created populations. This can happen if the contribution of a given population to AEPs is considerably less than the contribution of the remaining populations.

The first activity of AEP after having been created is to initiate NDM representing ANN without interneuron connections. Initially, all NDMs contain a minimal

acceptable number of rows and columns. This means that all ANNs created at the beginning of the evolution consist of only input and output neurons. Afterwards, if all AEPs created over an assumed period are not able to generate any satisfactory solution, NDMs are expanded by one row and one column which correspond to expansion of ANNs by one neuron. The process of growth of NDMs continues up to the moment when all of them represent ANNs of a maximal acceptable number of neurons.

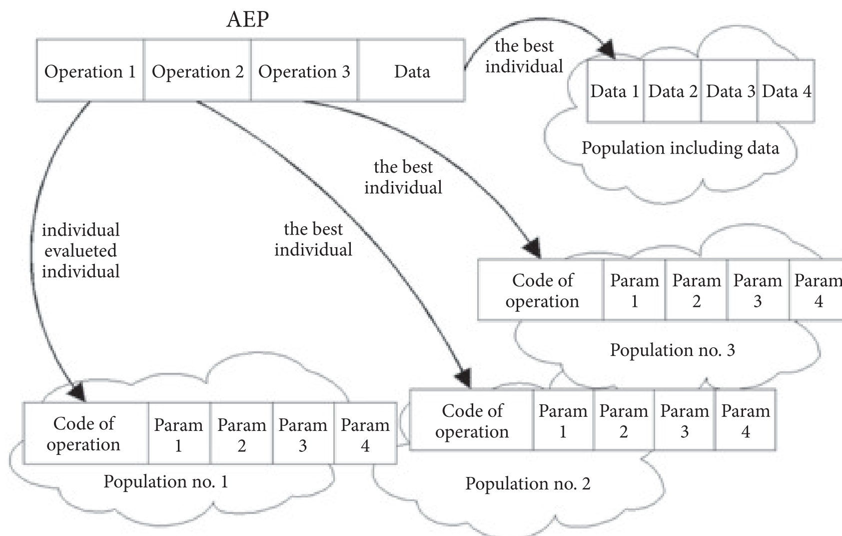


Fig. 5. AEP encoding scheme

Individual operations in AE can be encoded in two ways. For example, `CHGC0` presented in Fig. 2 is encoded in the form of binary string including five blocks of genes. The first block determines a code of the operation (e.g. binary 00000 indicates that we deal with `CHGC0`), while the remaining blocks contain the binary representation of four parameters of the operation.

`CHG_#2` (Fig. 3) is represented in a somewhat different way. The encoded form of this operation resembles classifier from Learning Classifier Systems [1, 3]. Similarity between classifier and the encoded operation results from the use of the so-called *don't care* symbol ("#") in both cases. Each encoded `CHG_#` consists of four blocks of genes. The first single-bit block determines one of two possible variants of the operation (`CHG_#1`, `CHG_#2`, see Appendix 1). The second and the third block indicate location of changes performed by the operation (*don't care* symbol is used for this purpose). The last block specifies the value of the integer parameter of the operation. The example use of *don't care* symbol to locate changes in NDM is illustrated in Fig. 6.

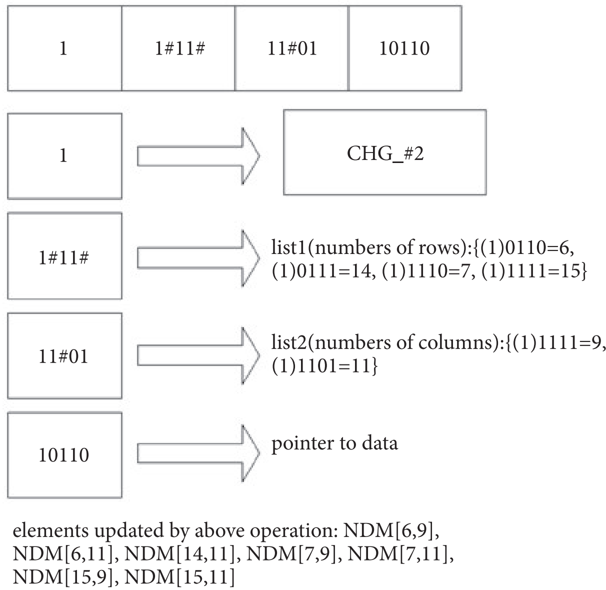


Fig. 6. Encoding `CHG_#2` operation ("#" is interpreted as either 0 or 1)

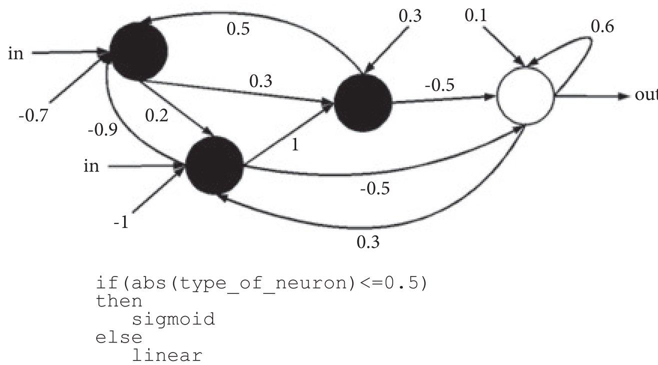
3. Using AE to form dynamic ANNs

In the previous experiments [10, 11, 12], AE was used to form ANNs with constant, invariable architecture. To form such ANNs, NDMs took the form of the classical CM. The construction of CM is presented in Fig. 7. The basic part of the matrix includes synaptic weights of interneuron connections, which do not change over time. Apart from the basic part, the matrix also contains additional columns describing parameters of neurons, e.g. type of neuron (sigmoid, radial), bias etc.

NDM used to represent a dynamic ANN has somewhat different structure from that of CM described above. Each NDM, in addition to a topology of ANN, also defines a process of changes occurring in the network during its lifetime. In total, NDM includes N rows and $Z = 2M + 2$ columns (Fig. 8) where N denotes the number of hidden and output neurons, whereas M is the number of all neurons in ANN. As before, extra two columns include additional information about neurons.

The main part of NDM used to define a dynamic ANN consists of two sub-matrices of equal size ($N \times M$). The first sub-matrix determines the topology of ANN, i.e. it indicates which connections exist in ANN and which do not. Each element of this sub-matrix unequal to zero informs about a connection between neurons. The sign of this element determines the sign of the connection, while the value of the element determines the type of Hebb rule assigned to the connection. For example, value –0.2 (it is assumed that all elements of NDM are from the range

	input neuron	input neuron		output neuron	bias	type of neuron
input neuron	0	0.2	0.3	0	-0.7	0.1
input neuron	-0.9	0	1	-0.5	-1	0.9
	0.5	0	0	-0.5	0.3	0.2
output neuron	0	0.3	0	0.6	0.1	0.5




```
if(abs(type_of_neuron)<=0.5)
then
   sigmoid
else
   linear
```

Fig. 7. NDM as Connectivity Matrix

	input neuron	input neuron		output neuron	input neuron	input neuron		output neuron	bias	parameter of neuron
	-0.1	0.5	0.6	-0.5	0.3	0.2	0.9	-0.4	-0.3	0.9
input neuron	-0.6	0.3	-0.1	0.6	0.1	0.5	-0.9	0.6	-0.2	0.3
	assignment of learning rules				learning rates					

Fig. 8. NDM used to create dynamic ANN

<–1, 1>) of element $NDM[n, m]$ ($n = 1...N$, $m = 1...M$, neurons are indexed from 0 to M) informs about both the negative connection between m^{th} and $[n + (M – N)]^{th}$ neuron (there are not connections between input neurons) and a plain Hebb rule assigned to this connection. In the experiments described further, five types of Hebb rules were used [2, 14]:

(1) *Plain Hebb rule*: can only strengthen the synapse proportionally to the correlated activity of the pre- and post-synaptic neurons

$$\Delta w = (1 - w)xy, \tag{1}$$

where is the synaptic weight, corresponds to change in the weight w, and x, y are respectively presynaptic and postsynaptic activity of neuron.

(2) *Postsynaptic rule*: behaves as the plain Hebb rule, but in addition it weakens the synapse when the postsynaptic node is active, and the presynaptic is not

$$\Delta w = w(-1 + x)y + (1 - w)xy. \tag{2}$$

(3) *Presynaptic rule*: weakens the synapse when the presynaptic unit is active but the postsynaptic is not

$$\Delta w = wx(-1 + y)y + (1 - w)xy. \tag{3}$$

(4) *Covariance rule*: strengthens the synapse whenever the difference between the activations of the two neurons is less than half their maximum activity, otherwise the synapse is weakened. In other words, this rule makes the synapse stronger when the two neurons have similar activity and makes it weaker when they do not

$$\Delta w = \begin{cases} (1 - w)\Psi(x, y) \text{ if } \Psi(x, y) > 0 \\ w\Psi(x, y) \text{ otherwise,} \end{cases} \tag{4}$$

where $\Psi(x, y) = \tanh(4(1 - |x - y|) - 2)$ is a measure of a difference between presynaptic and postsynaptic activity. $\Psi(x, y) > 0$ if the difference is higher than or equal to 0.5 and $\Psi(x, y) < 0$ if the difference is smaller than 0.5.

(5) *"Zero" rule*

$$\Delta w = 0. \tag{5}$$

Each Hebb rule presented above corresponds to a different value in NDM.

The second sub-matrix of NDM includes learning rates necessary to update the strength of each synaptic weight. For example, $NDM[n, m] = -0.2$, where $n = 1...N$ and $m = M...2M$, informs that the learning rate used to update the connection between $[m - M]^{th}$ and $[n + (M - N)]^{th}$ neuron amount to $|-0.2|$. If a connection between neurons exists but the learning rate corresponding to this connection amount to zero, to update the strength of the connection, a default nonzero value of the learning rate is used.

The Hebb rules from the first part of NDM and the learning rates from its second part are necessary to determine changes that take place in each interneuron connection. Each synaptic weight in ANN alters according to the following formula:

$$w_{ij}^{t} = w_{ij}^{t-1} + \eta_{ij}\Delta w_{ij}, \tag{6}$$

where w_{ij}^{t}, w_{ij}^{t-1} are synaptic weights between the jth and the ith neuron, respectively, after and before update and $0 \le \eta_{ij} \le 1$ is the learning rate.

First, once ANN is created, all weights of all nonzero connections are fixed in some assumed manner, for example at random. Then, synaptic weights change according to formula (6). All synapses can change the strength but they cannot change the sign, which is determined permanently in NDM. The synaptic strength cannot grow indefinitely. All weights are from the range <0, 1>. This is possible thanks to self-limiting mechanism in all the Hebb rules specified above. An update of each synaptic weight occurs once an input signal is propagated to output neurons, i.e. each time a decision has been taken by ANN.

4. Experiments

The main goal of the experiments was to test whether AE can be used to create simple dynamic, self-organizing ANNs. In the experiments, the task of all ANNs was to control a set of cooperating predators whose common goal was to capture a fast moving prey. The experiments were performed in a configuration with one prey and three chasing predators. The prey behaved according to a simple deterministic strategy, whereas the predators were controlled by a single ANN. In the experiments, the following variants of AE were tested: AEPs consisting of binary encoded operations (AEPs^{01}, e.g. `CHGC0`), AEPs including `CHG_#` operations (AEPs$^{\#}$), and AEPs containing both types of operations (AEPs$^{01\#}$).

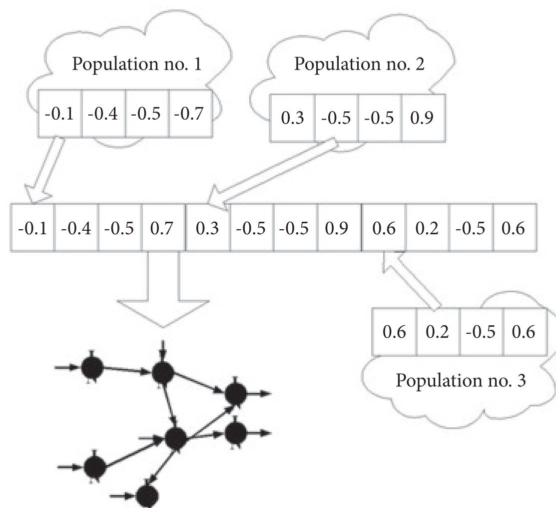


Fig. 9. Co-evolutionary version of SNE used in experiments

Apart from AE, for the purpose of comparison, a modified version of SNE was also used in the experiments. SNE means here the method in which all the information necessary to create a network is stored in a single chromosome. In the case of the dynamic ANNs, the chromosome mentioned has to include genes assigning the Hebb rules and the learning rates to connections and parameters to neurons. All the chromosomes encoding ANNs are grouped in a single population. In the modified version of SNE, used in the experiments, the dynamic ANNs evolved according to the scheme proposed by Potter and De Jong, i.e. in a similar way as AEPs in AE. Each chromosome encoding ANN were divided into three separate parts of more or less the same length (the number of the parts were determined arbitrary by the author) and each part evolved in a separate population.

4.1. Environment

The predators and the prey lived in a common environment. To represent the environment, 20 × 20 square was used. The square was without any obstacles but with two barriers located on its left and right-hand side. Both barriers caused the predators as well as the prey to move right or left only to the point at which they reached one of the barriers. Attempts to move further in the direction of the barrier ended up in failure. In order to ensure infinite space for the predators and the prey and for their struggles, the environment was opened at the bottom and at the top. This means that every attempt of movement beyond upper or lower border of the square caused the object, making such an attempt to move to the opposite side of the environment. As a result, the simple strategy of predators, consisting in chasing the prey, did not work. In such situation, the prey, in order to evade predators, could simply escape upwards or downwards.

4.2. Residents of the artificial world

In the experiments, three predators and one prey coexisted in the artificial environment. The predators were controlled by ANN produced by AEP. They could select five actions: to move in North, South, West, East direction or to stand still. The length of the step made by each predator was 1, while the step made by the prey amounted either to 2 or to 1. In order to capture the prey, the predators had to cooperate. Their speed was either two times lower or the same as the speed of the escaping prey so they could not simply chase the prey to capture it. The prey was captured if the distance between it and the nearest predator was lower than 2.

In the experiments, the predators could see the whole environment. The predators based on the decision which actions to select on the prey's relative location with reference to each of them. In order to perform the task, ANN controlling the predators had to possess six inputs and three outputs. The outputs provided decisions to the

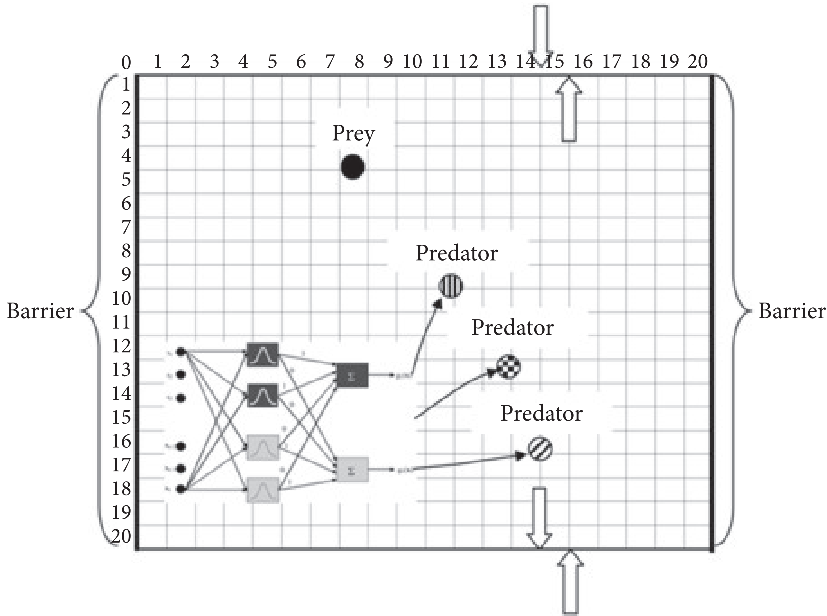


Fig. 10. Artificial world in which task of predators was to capture prey

predators whereas the inputs informed them about prey's location in relation to each of them.

In the experiments, two types of prey were used, i.e. a simple prey and an advanced prey. As for the simple prey, it was controlled by a simple algorithm which forced it to move directly away from the nearest predator but solely in the situation when the distance between it and the nearest predator was lower than or equal to 5. In the remaining cases, i.e. when no predator was closer to the prey than the assumed distance, the prey did not move. In the situation when the selected prey's action could cause hitting the barrier, another move was chosen. Alternative move prevented from hitting the wall, and at the same time, it maximally increased the distance between the prey and the nearest predator. The prey, when was running away could select four actions: to move in North, South, West or East direction. A strategy of the simple prey is presented below:

$$\pi(s)=\begin{cases} \text{StandStill if } \underset{p\in P}{\forall} d(p,s)>5 \\ \arg\max_{a\in A} D\left(s,a,\arg\min_{p\in P} d(p,s)\right) \text{ otherwise,} \end{cases} \tag{7}$$

where: P – is the set of predators;

A – is the set of actions of prey (A = {StandStill,North,South,West,East});

$d(p, s)$ – is the distance between prey and the predator p in state of the environment s;
$D(s, a, p)$ – is the distance between prey and the predator p in state of environment which is direct consequence of the action a performed by prey in the state s.

Making decision, the advanced prey, unlike its simpler counterpart, always took into consideration the location of all predators situated close to it. Actions performed by the advanced prey always maximized the average distance between the prey and all the predators that were close to it. Other aspects of behaviour of the advanced prey, i.e. behaviour near the barrier, behaviour away from the predators, and actions which the prey could perform in each step, were the same as in the case of the simple prey. A strategy of the advanced prey is presented below:

$$\pi(s) = \begin{cases} \text{StandStill if } \underset{p \in P}{\forall} d(p,s) > 5 \\ \underset{a \in A}{\arg\max} \left(\frac{1}{P_5(s)} \sum_{p \in P_5(s)} D(s,a,p) \right) \text{otherwise,} \end{cases} \tag{8}$$

where: $P_5(s) = \{p \in P, d(p,s) \leq 5\}$.

4.3. Neural controllers

In the experiments, feed-forward ANNs (FFANN) with Hebb self-organization and recurrent ANNs (RANN) with Hebb self-organization were tested. All ANNs had six inputs and three outputs. The number of outputs corresponded to the number of predators. In turn, the number of inputs was twice the number of predators. Each output gave commands to one predator. In turn, each input informed about vertical or horizontal distance between the prey and one of the predators.

In the experiments, ANNs could grow only to a certain limit. At first, networks including minimal acceptable number of neurons were created. If the evolution could not produce any successful ANN within some assumed period, all ANNs were expanded by one neuron. This procedure was continued up to the moment when all ANNs included 15 neurons, i.e., 6 input neurons, 6 hidden and 3 output neurons. In the case when even fifty-neuron ANNs could not grasp the prey in all investigated situations, the evolutionary process was stopped.

4.4. Parameters of evolutionary process

In all the experiments with SNE, Canonical GA (CGA) [3] was used. The same algorithm was also applied, in the experiments with AE, to form data for AEPs. Operations for AEPs were produced by means of Eugenic Algorithm (EuA) [13].

All chromosomes used in the experiments consisted of 7-bit blocks of genes. Each chromosome-operation consisted either of five blocks of binary genes (one block for code of an operation and the remaining four blocks for parameters of the operation, e.g. `CHGC0`) or of four blocks of genes including zeros, ones and *don't cares* (one block for code of an operation, two blocks for encoded lists and the last forth block for integer parameter of the operation, e.g. `CHG_#1`). The list of applied operations is presented at the end of the paper (Appendix 1). The length of chromosomes from SNE was dependent on the size of ANN. For example, ANN consisting of 12 neurons was represented in the form of three chromosomes consisting of fifty two blocks of binary genes. Chromosomes-data could change the length during consecutive co-evolutionary cycles. In order to make such a change possible, CGA in addition to crossover and mutation used a cut-splice operator. The implementation of crossover, used in the experiments, always produced offspring of the same length as parents. The cut-splice operator, which was always activated after crossover and mutation, modified the size of a chromosome through adding or removing a single block of genes (single data) from the same end of the chromosome. In the experiments, the chromosomes-data could maximally contain 20 data, i.e. 20 7-bit blocks of genes. Each use of an excessive number of data caused drastic decrease in fitness of the AEP. In the experiments, AEPs could include 12 operations. Initially, each AEP contained one operation and one set of data from two different populations. Consecutive populations with operations were added every 5000 of co-evolutionary cycles if generated AEPs were not able to achieve progress in performance within this period. Populations with operations and data could be also replaced by newly created populations when the contribution of a substituted population to created AEPs was considerably less than the contribution of the remaining populations. The same procedure was also applied in SNE. The contribution of a population was measured as average fitness of individuals belonging to that population. The remaining values essential for the experiments are the following:

- size of each population: 20 individuals (AE), 40 (SNE),
- number of co-evolutionary cycles for one fixed structure of ANN: 50 000 (afterwards, all ANNs were expanded by one neuron and evolutionary process started again),

Parameters of CGA:

- crossover probability: 0.7,
- per-bit mutation probability: 0.01,
- cut-splice probability: 0.1 (in chromosomes-data),

Parameters of EuA:

- selection noise: 0.01, 0.2,
- creation rate: 0.01, 0.2,
- restriction operator: on.

4.5. Evaluation process

In order to evaluate ANNs ten different scenarios were used. At first, each ANN was tested in the scenario no. 1. If the predators controlled by ANN could not capture the prey during an assumed period, the test was stopped and the network received an appropriate evaluation. The evaluation depended on the distance between the prey and the nearest predator. However, if the predators grasped the prey, ANN was run in a next scenario. In the experiments, the predators could perform 100 steps before the scenario was interrupted.

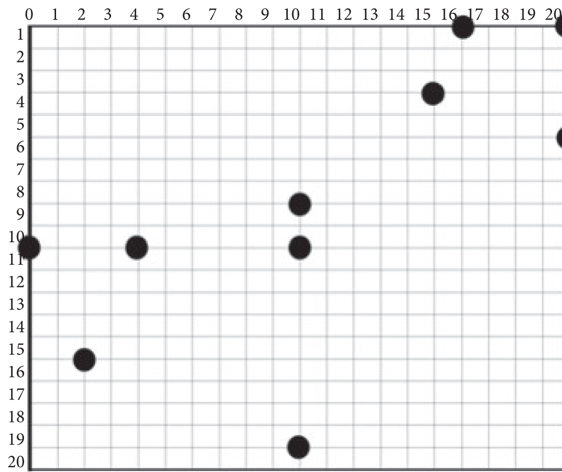


Fig. 11. Initial locations of prey

The scenarios used in the experiments differed in the initial position of the prey, in the length of step of the prey and in the type of the prey applied (simple or advanced). Consecutive scenarios were more and more difficult. At first, the predators had to capture the simple prey that was as fast as them. The predators, which passed the first exam, had to pit against the simple prey that was twice faster than them. In the next step, the speed of the prey was decreased once again. However, this time the predators had to face the advanced prey which took better decisions than its predecessor. In the last stage, the predators which coped with all earlier scenarios had to capture the advanced, fast prey. In all the scenarios, starting positions for all three predators were the same. The predators always started from position (0,0). Below, described are all eight scenarios:

- Scenario no 1: simple prey (20,5), step of prey = 1,
- Scenario no 2: simple prey(10,8), step of prey = 1,
- Scenario no 3: simple prey (15,3), step of prey = 2,
- Scenario no 4: simple prey (0,10), step of prey = 2,

- Scenario no 5: advanced prey (16,0), step of prey = 1,
- Scenario no 6: advanced prey (2,15), step of prey = 1,
- Scenario no 7: advanced prey (10,19), step of prey = 2,
- Scenario no 8: advanced prey (4,10), step of prey = 2,
- Scenario no 9: advanced prey (10,10), step of prey = 2,
- Scenario no 10: advanced prey (20,0), step of prey = 2.

To evaluate ANNs, the following fitness function was used:

$$f\left(ANN\right)=\sum_{i=1}^{n} f_i, \tag{9}$$

$$f_i = \begin{cases} d_{\max} - \min_{p \in P} d\left(p, s_{100}^{i}\right) \text{ prey not captured in } i^{\text{th}} \text{ scenario} \\ f_{captured} + \left(100 - m_i\right)/a \text{ prey captured in } i^{\text{th}} \text{ scenario} \\ 0 \text{ prey not captured in the previous scenario,} \end{cases} \tag{10}$$

where: f_i is the reward received in the ith scenario;
d_{max} is the maximum distance between two points in environment;
s^{i}_{100} is the end state in i^{th} scenario;
$f_{captured}$ is the reward for grasping prey in single scenario (in the experiments $f_{captured} = 100$);
m_i is the number of steps which predators needed to capture prey ($m_i < 100$);
a – this value prevents situation in which partial success is better than success in all scenarios;
n is the number of scenarios.

The total fitness of each ANN is a sum of rewards from the scenarios in which the network has taken part. The reward for a scenario depends on the chase result. In the case of success, ANN obtains an extra fitness for grasping the prey and additionally a reward reversely proportional to the number of steps which the predators had to make to capture the prey. In the case of failure, ANN obtains fitness proportional to the distance between the prey and the nearest predator.

4.6. Experimental results

30 evolutionary runs were performed for each ANN encoding method and for each type of ANN. The results of the experiments are presented in Table 1. For the purpose of comparison, the results of the previous experiments with the static ANNs [10], are also included in the table (the previous experiments were carried out in the same conditions as the experiments reported in the paper).

Generally, the experiments showed that AE can be successfully used to create the dynamic ANNs. However, they also showed that the dynamic ANNs are considerably much harder to generate than the static ones. As it turned out, the only successful dynamic ANNs* were produced by means of AEPs^{01}.

TABLE 1

Experimental results; column (1)-average fitness (best fitness); column (2)-% of successful ANNs; column (3)-average number of neurons in successful ANN (minimal number of neurons); column (4)-average length of successful AEP, number of neurons+number of data (shortest AEP); column (5)-average number of co-evolutionary cycles necessary to generate successful AEP (minimal number); e.g. fitness = 850 means that prey was captured in eight scenarios; results of the previous experiments are marked in grey

	(1)	(2)	(3)	(4)	(5)
static FFANN (AEP^{01})	1028.14 (1069.75)	93%	11.5 (10)	4.9 + 13.8 (2 + 21)	196834.3 (6614)
static FFANN (AEP$^{\#}$)	915.24 (1081.76)	73%	14.6 (12)	6.2 + 15.4 (3 + 14)	335692.4 (158808)
static FFANN (AEP$^{01\#}$)	998.98 (1088.65)	80%	13.2 (11)	5 + 13.3 (3 + 12)	286951.7 (108456)
dynamic FFANN (AEP^{01})	836.79 (1055.66)	28%	14.7 (12)	5.8 + 14.5 (5 + 9)	347053.8 (179808)
dynamic FFANN (AEP$^{\#}$)	480.41 (563.38)	0%			
dynamic FFANN (AEP$^{01\#}$)	546.1 (659.27)	0%			
dynamic FFANN (SNE)	706.78 (954.12)	0%			
dynamic RANN (AEP^{01})	728.07 (1065.33)	21%	14.5 (12)	5.7 + 15.7 (5 + 14)	339651.4 (183421)
dynamic RANN (AEP$^{\#}$)	362,04 (441.32)	0%			
dynamic RANN (AEP$^{01\#}$)	489.59 (648.32)	0%			
dynamic RANN (SNE)	621,65 (893.33)	0%			

Only 28% dynamic FFANNs and 21% dynamic RANNs were successful in this case. It is three times worse result compared to the same method used to create static FFANNs. The remaining encoding methods did not generate even one ANN

* ANN which captured the prey in all tested scenarios.

effective in all ten scenarios. The most problems with generating effective ANNs had AEPs$^{\#}$. In most cases, ANNs produced by AEPs$^{\#}$ had difficulties even with scenarios in which the simple prey was used. Creating the static ANNs was much easier for AEPs$^{\#}$. 73% static FFANNs created by means of AEPs$^{\#}$ were successful. Even though AEPs$^{01\#}$ generated somewhat better dynamic ANNs than AEPs$^{\#}$, all of them were significantly less effective than ANNs produced by AEPs^{01}. As before, AEPs$^{01\#}$ were much more effective in creating the static ANNs. 80% static FFANNs created by means of AEPs$^{01\#}$ were successful. SNE appeared to be better solution than AEPs$^{\#}$ and AEPs$^{01\#}$. Nevertheless, most dynamic ANNs generated by means of SNE were strongly unsatisfactory. With regard to the successful ANNs, generated during the tests, most of them included maximal acceptable number of neurons, i.e. 15 neurons. The smallest successful dynamic ANNs contained 12 neurons.

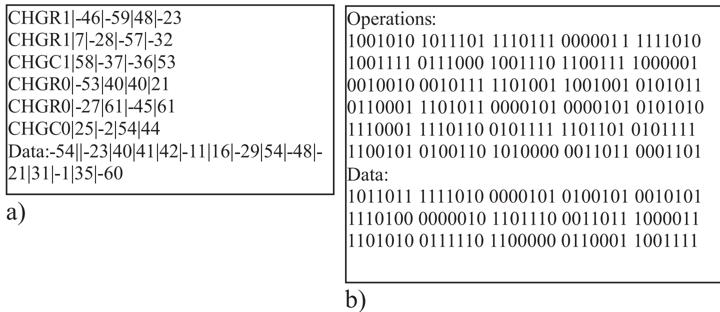


Fig. 12. (a) Example AEP^{01} which created successful dynamical FFANN, (b) encoded form of AEP presented in point (a)

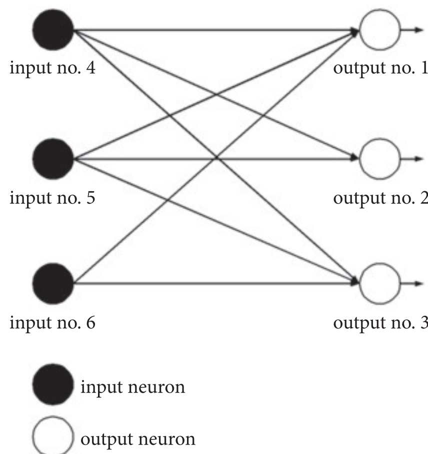


Fig. 13. Successful dynamic FFANN created by AEP presented in Fig. 12 after removing all unnecessary neurons and connections (ANN without modifications included 12 neurons)

In most cases, ANNs used only two predators to grasp the prey. The third predator usually moved in one direction and did not take part in the chase. The first activity to capture the prey was always to press the prey against one of the barriers. In the second phase of the hunt, one predator usually patrolled a sector of the environment near the barrier walking forth and back, right and left. It was careful enough not to touch the barrier. At the same time, the second predator moved along the barrier and forced the prey to escape in the direction of the first predator.

```
CHGR1|46|-25|51|-41
CHGR1|-60|21|-2|12
CHG0|-50|-51|4
CHGR1|-54|59|55|59
CHGM0|8|50|-35|-22
Data:-16|-23|-37|-31|60|-46|62|43|5|-30|43|-24|35|12
```

a)

```
Operations:
0001011 0011101 1100110 0110011 1100101
0001100 1001111 0101010 1010000 0001100
0000001 1010011 1110011 0001000 0100111
1001100 1011011 0110111 0111011 0110111
1101001 0000100 0010011 1110001 1011010
Data:
1000010 1111010 1101001 1111110 0001111
1011101 0011111 0110101 0101000 1011110
0110101 1000110 0110001 0001100
```

b)

```
0.19 0.98 0.55 -0.73 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -0.36 0.68
-0.36 0.68 0.19 0.98 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.730.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73 0.73-0.58 0.07
-0.47 -0.58 0.07 -0.36 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.68 -0.49
0.68 -0.49 -0.47 -0.58 -0.79 0 0 0 0 0 0 0 -0.95 -0.95 -0.95 0 0 0 0 0 0 0 0 0 -0.38 0.95
-0.38 0.95 0.68 -0.49 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.55 -0.73
0.55 -0.73 -0.38 0.95 0 0 0 -0.85 -0.85 -0.85 -0.85 0 0 0 0 0 0 0 0 0 0 0 0 0 0.19 0.98
```

c)

Fig. 14. (a) Example AEP^{01} which created successful dynamic RANN; (b) encoded form of AEP presented in point (a); (c) NDM generated by AEP presented in point (a) and (b)

Fig. 15. (a) example behaviour of predators and prey in scenario no. 7 (neuro-controller: FFANN presented in Fig. 13); (b) example behaviour of predators and prey in scenario no. 8 (neuro-controller: RANN whose NDM is presented in Fig. 14c). Circles indicate initial positions of predators and prey (black circle – prey, circle with vertical stripes – predators), round symbols with diagonal lines denote final positions, arrowed lines indicate directions of movement (solid line – prey, dashed or dotted lines – predators) whereas black boxes determine time of occurrence of individuals in a given place

In the second stage of the chase, i.e. after pressing the prey against the barrier, sometimes we also observed other strategy of the predators than the one described above. In the second of the strategies, the predator waiting for the prey lurked near the same barrier. In turn, the second predator or as it is depicted in Fig. 15 the second and the third predators moved along the barrier but at some distance from it. They forced the prey to escape in the direction of the first of the predators mentioned, i.e., in the direction of the predator waiting on the opposite side of the environment. Both strategies of the predators presented above seem very simple, nevertheless, as the experiments showed they also turned out to be very effective.

5. Summary

The paper presents usage of AE to create the dynamic ANNs with Hebb self-organization. In order to find out capabilities of AE to create such ANNs, experiments in the predator-prey problem300

were carried out. In the experiments, the task of ANNs was to supervise a group of predators whose common goal was to capture a fast moving prey behaving according to a simple strategy. In the experiments, three variants of AE were used, i.e. $AEPs^{01}$, $AEPs^{\#}$, and $AEPs^{01\#}$. Moreover, the method called standard neuro-evolution was also tested. The research showed that AE is able to create simple dynamic ANNs. However, creating such ANNs appeared to be considerably harder than generating the static ANNs. $AEPs^{01}$ turned out to be a variant of AE which generated the most effective the dynamic ANNs. The remaining methods tested in the experiments were significantly less effective.

Artykuł wpłynął do redakcji 19.04.2010 r. Zweryfikowaną wersję po recenzji otrzymano w listopadzie 2010 r.

REFERENCES

[1] M. Butz, *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*, IlliGAL Report, no. 2004034, University of Illinois, 2004.

[2] D. Floreano, J. Urzelai, *Evolutionary robots with online self-organization and behavioural fitness*, Neural Networks, 13, 2000, 431-443.

[3] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison Wesley, Reading, Massachusetts, 1989.

[4] F. Gruau, *Neural network Synthesis Using Cellular Encoding and The Genetic Algorithm*, PhD Thesis, Ecole Normale Superieure de Lyon, 1994.

[5] S. Luke, L. Spector, *Evolving Graphs and Networks with Edge Encoding: Preliminary Report*, Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28-31, Stanford University, 1996, 117-124.

[6] G. F. Miller, P. M. Todd, S. U. Hegde, *Designing Neural Networks Using Genetic Algorithms*, Proceedings of the Third International Conference on Genetic Algorithms, 1989, 379-384.

[7] P. Nordin, W. Banzhaf, F. Francone, *Efficient Evolution of Machine Code for {CISC} Architectures using Blocks and Homologous Crossover*, Advances in Genetic Programming III, MIT Press, 1999, 275-299.

[8] M. Potter, *The Design and Analysis of a Computational Model of Cooperative Coevolution*, PhD thesis, George Mason University, 1997.

[9] M. A. Potter, K. A. De Jong, *Cooperative coevolution: An architecture for evolving coadapted subcomponents*, Evolutionary Computation, 8(1), 2000, 1-29.

[10] T. Praczyk, *Evolving co-adapted subcomponents in assembler encoding*, International Journal of Applied Mathematics and Computer Science, 17(4), 2007, 549-563.

[11] T. Praczyk, *Modular networks in Assembler Encoding*, Computational Methods in Science and Technology, CMST 14(1), 2008, 27-38.

[12] T. Praczyk, *Using assembler encoding to solve the inverted pendulum problem*, Computing and Informatics (in press)

[13] J. W. Prior, *Eugenic Evolution for Combinatorial Optimization*, Master's thesis TR AI98-268, The University of Texas at Austin, 1998.

[14] J. Urzelai, D. Floreano, *Evolution of Adaptive Synapses: Robots with Fast Adaptive Behaviour in New Environments*, Evolutionary Computation, 9(4), 2001, 495-524.

A list of operations used in the experiments

Binary encoded operations:

CHG – Update of an element. Both the new value and address of the element are located in parameters of the operation.

CHGC0 – see page 3.

CHGC1 – Update of a certain number of elements in a column. An index of the column, an index of the first element in the column, the number of changed elements and a new value for the column's elements, the same for all elements, are located in parameters of the operation.

CHGC2 – Update of a certain number of elements in a column. A new value for each element is a sum of the operation's parameter and the current value of the element. The second parameter of the operation is an index of the column. The third and fourth parameters of the operation determine the number of changed elements and index of the first element in the column, respectively.

CHGC3 – A number of elements from one column are transferred to another column. Both columns are indicated by parameters of the operation. The number of transferred elements and index of the first element in the source column are also included in parameters of the operation.

CHGC4 – Update of a certain number of elements in a column. A new value for each element is a sum of the current value of the element and a value from data part of AEP. An index of the column, an index of the first element in the column, the number of changed elements, and a pointer to data, where ingredients of individual sums are memorized, are located in parameters of the operation.

CHGR0 – like **CHGC0** but an update refers to a row of NDM.
CHGR1 – like **CHGC1**.
CHGR2 – like **CHGC2**.
CHGR3 – like **CHGC3**.
CHGR4 – like **CHGC4**.
CHGM0 – Change of a block of elements. The elements are updated in columns, in turn, one after another, starting from an element pointed by parameters of the operation. The number of changed elements and a place in the memory where new values for the elements are located are determined by parameters of the operation.
CHGM1 – like **CHGM0,** but a new value for each element is a sum of its current value and parameter of the operation.
CHGM2 – like **CHGM0,** but a new value for each element is a sum of its current value and a value from the data part of AEP. The number of changed elements and a place in the memory where arguments of individual sums are located – are determined by parameters of the operation.
JMP – Jump operation. The number of jumps, a pointer to a next operation and new values for the registers are located in parameters of the operation.

Operations encoded in the form of strings including zeros, ones and *don't cares*:
CHG_#1 – Update of a set of elements in NDM. All the updated elements have the same value.
CHG_#2 – see page 3.

T. PRACZYK

Tworzenie sieci neuronowych z samoorganizacją Hebba za pomocą Kodowania Asemblerowego

Streszczenie. Kodowanie Asemblerowe jest metodą neuro-ewolucyjną, która do tej pory stosowana była wyłącznie do konstrukcji sieci neuronowych o stałej, nie zmiennej w czasie architekturze. W celu sprawdzenia skuteczności metody w tworzeniu innych typów sieci neuronowych, przeprowadzono szereg eksperymentów. W ich trakcie zadaniem Kodowania Asemblerowego była konstrukcja rekurencyjnych oraz jednokierunkowych sieci neuronowych z samoorganizacją Hebba. Tworzone sieci wykorzystywane były do sterowania zespołem „drapieżców", których celem było pochwycenie szybko poruszającej się „ofiary" (predator-prey problem) postępującej zgodnie z pewną prostą strategią. Wyniki Kodowania Asemblerowego uzyskane w trakcie badań porównano z osiągami innej metody neuro-ewolucyjnej.
Słowa kluczowe: sieci neuronowe, neuro-ewolucja, problem drapieżca–ofiara