Method for Creating a Computer Agent Based on the Jordan-Elman Neural Network for Supply Chains

Eugene Fedorov[a], Olga Nechyporenko[a] and Tetiana Neskorodieva[b]

[a]Cherkasy State Technological University, Shevchenko blvd., 460, Cherkasy, 18006, Ukraine
[b]Vasyl' Stus Donetsk National University, 600-richchia str., 21, Vinnytsia, 21021, Ukraine

Abstract

The paper proposes a method for creating a computer agent for supply chains. The novelty of the research lies in the fact that to increase the efficiency of the computer agent, its functioning is based on the connectionist approach instead of using the classical production and logical approach. To expand the range of tasks solved by agents, the article proposes a reactive agent with feedback, which makes a decision based on perception or a sequence of perceptions and a previous action or a sequence of previous actions, as well as a reactive agent with an internal state and feedback, which is an extension of the reactive agent with an internal state and makes a decision based on perception, previous internal state, and previous action. For a reactive agent with an internal state and feedback, a Jordan-Elman artificial neural network was proposed, which is a combination of Jordan and Elman neural networks, and the structure of its model was determined in the course of a numerical study. The experiments performed showed that when the number of hidden neurons is not less than the number of neurons in the input layer, the value of the root mean square error does not change significantly, and the selected network gives results with a minimum error. Methods for determining the parameters of the proposed Jordan-Elman neural network model were proposed. This made it possible to ensure high speed and accuracy of calculations based on the model. The proposed method for creating an agent based on artificial neural networks can be used in various intelligent computer systems that use multi-agent interaction.

Keywords

supply chain, multi-agent system, Jordan-Elman artificial neural network, agent functioning models, Adam method, charged system search method

1. Introduction

The fourth industrial revolution or Industry 4.0 has brought about rapid changes in technology, manufacturing and social processes in the 21st century due to increasing interconnection and intelligent automation. Part of this phase of industrial change is the integration of artificial intelligence with robotics, which blurs the boundaries between the physical, digital and biological worlds. In terms of logistics, Industry 4.0 is expressed in high customer orientation, strong links between industries, dynamic supply chains. These characteristics can be achieved through the use of multi-agent systems.

The work aims to develop an agent based on an artificial neural network. To achieve the goal, the following tasks were set and solved:

- propose a formal description of reactive agents' functioning models;
- propose a formal description of neural networks-based reactive agents' functioning models;
- create a neural network model for the proposed agent with internal state and feedback;

IntelITSIS'2023: 4th International Workshop on Intelligent Information Technologies and Systems of Information Security, March 22–24, 2023, Khmelnytskyi, Ukraine
EMAIL:fedorovee75@ukr.net (E. Fedorov); olne@ukr.net (O. Nechyporenko); t.neskorodieva@donnu.edu.ua (T. Neskorodieva)
ORCID: 0000-0003-3841-7373 (E. Fedorov); 0000-0002-3954-3796 (O. Nechyporenko); 0000-0003-2474-7697 (T. Neskorodieva)

- propose a performance assessment criterion of a neural network model for the proposed agent with an internal state and feedback;
- create methods for determining the parameters values of the neural network model for the proposed agent with internal state and feedback;
- perform numerical studies.

2. Literature review

Currently, the main types of computer agents in multi-agent systems are reactive and proactive agents.

Traditionally, a simple reactive agent decides by applying production rules (called behaviors), and this agent has a database (which stores its current state) and a production rule base. The condition of a production rule is a perception (or a sequence of perceptions), the conclusion is an action.

Advantages of simple reactive agents [1]:

1. Ease of implementation.
2. Quick reaction (quick decision-making).
3. Robust decision-making.
4. Ease of organization of multi-agent interaction.

Disadvantages of simple reactive agents [2]:

1. Simple reactive agents do not use world models, so they must have enough local information (i.e., information about their current state) to determine an acceptable action.
2. Since simple reactive agents make decisions based on local information (i.e., information about their current state), it is difficult for them to make a decision based on non-local information (i.e., information about the current state of other agents).
3. Weak learning ability of simple reactive agents.
4. It is not clear how the resulting behavior of simple reactive agents emerges from their interactions between themselves and the environment since inference is not used.
5. It is difficult to build a simple reactive agent with a large number of production rules.
6. Low level of intelligence, providing low autonomy.

Traditionally, a reactive agent with an internal state (or based on a model) makes a decision through inference, and has a database (it stores information about the state of the world - an internal state), a world model (a knowledge base containing knowledge about how the world changes independently from the agent, and knowledge about how the agent's actions affect the world) and the inference engine.

Advantages of reactive agents with internal state [3]:

1. Simple logical semantics.
2. High level of intelligence, providing high autonomy.

Disadvantages of reactive agents with internal state [4]:

1. The environment can change faster than an agent with an internal state makes a decision.
2. The complexity of mapping the environment into symbolic (logical) perception (in the form of logical formulas) performed by the perception function. For example, there is the problem of converting an image or sound to a set of declarative statements representing that image or sound.
3. The complexity of representing the dynamic environment properties of the real world using classical first-order predicate logic. Representing even simple procedural knowledge (that is, knowing what to do) in traditional logic can be quite difficult.
4. The complexity of organizing multi-agent interaction.

Traditionally, a proactive agent decides on the choice of a goal (from possible goals) and how to achieve it (forms an action plan) based on a logical conclusion, and has a database (it stores information about the state of the world - an internal state, as well as a selected goal), a world model (a knowledge base containing knowledge about how the world changes independently of the agent, and knowledge about how the agent's actions affect the world) and an inference engine. A proactive agent may also use a utility function.

The advantages and disadvantages of proactive agents are analogous to the advantages and disadvantages of internal state reactive agents.

Thus, the lack of effectiveness of the considered computer agents is a relevant problem.

Nowadays, instead of expert systems with logical inference, used in decision-making agents, artificial neural networks are actively used [5]. Depending on the types of agents used and the tasks they solve, static, dynamic and recurrent neural networks can be selected [6, 7].

Advantages of neural networks:

- the possibility of their training and adaptation [8, 9];
- parallel information processing that increases computing power [10];
- the ability to identify patterns in the data, their generalization, i.e., extracting knowledge from data, so knowledge about the object is not required (for example, its mathematical model) [11,12].

Disadvantages of neural networks:

- a high probability of the training and adaptation method hitting a local extremum [13];
- inaccessibility for human understanding of the knowledge accumulated by the network (it is impossible to represent the relationship between input and output in the form of rules), since they are distributed among all of the elements of the neural network and are presented in the form of its weight coefficients [14, 15];
- difficulty in determining the structure of the network, since there are no algorithms for calculating the number of layers and neurons in each layer for specific applications [8, 16];
- difficulty in forming a representative sample [17, 18].

Thus, none of the networks satisfies all the criteria.

To improve the efficiency of determining the parameter values of neural network models, metaheuristic search is used instead of local search.

Advantages of metaheuristic methods [19]:

- combines heuristic methods with an efficient strategy;
- low probability of the method hitting a local extremum due to the use of random search.

Disadvantages of metaheuristic methods [20-22]:

- the method may not converge;
- the method is not very accurate;
- iteration number is not present when searching for a solution;
- there is only a generalized method structure or the method structure is focused on solving only a specific problem;
- real potential solutions are inadmissible;
- the method is not designed for conditional optimization;
- there is no formalized search strategy for parameter values.

This raises the problem of constructing an effective metaheuristic optimization method for training neural networks.

3. Formal description of the reactive agents' functioning models

A simple reactive agent functioning model

Perception function

$$see: E \to Per$$

maps the current state of the environment into a new perception.

Action selection function

$$action: Per \to Ac$$

maps a new perception into a new action

or

$$action: Per^{*} \to Ac$$

maps a sequence of perceptions (new and previous) into a new action.

Functioning model of a reactive agent with an internal state

Perception function

$$see: E \to Per$$

maps the current state of the environment into a new perception.

State change function

$$next: I \times Per \to I$$

maps a previous internal state and a new perception into a new internal state.

Action selection function

$$action: I \to Ac$$

maps the new internal state into a new action.

In addition to these two types of reactive agents, this article proposes a feedback reactive agent that makes a decision based on a perception (or a sequence of perceptions) and an action (or a sequence of actions). Also, a reactive agent with an internal state and feedback is proposed, which is an extension of the reactive agent with an internal state and also considers action.

The functioning model of a reactive agent with feedback, proposed by the authors

Perception function

$$see: E \to Per$$

maps the current state of the environment into a new perception.

Action selection function

$$action: Per \times Ac \to Ac$$

maps a new perception and a previous action into a new action

or

$$action: Per^* \times Ac^* \to Ac$$

maps a sequence of perceptions (new and previous) and a sequence of previous actions into a new action.

The functioning model of a reactive agent with an internal state and feedback, proposed by the authors

Perception function

$$see: E \to Per$$

maps the current state of the environment into a new perception.

State change function

$$next: I \times Per \times Ac \to I$$

maps a previous internal state, a new perception, and a previous action into a new internal state.

Action selection function

$$action: I \to Ac$$

maps the new internal state into a new action.

4. Formal description of the neural networks-based reactive agents functioning models

The following main functioning models of reactive agents based on shallow neural networks are possible:

1. For a simple reactive agent, the non-linear regressive/autoregressive model corresponds to:

- forward neural network (FNN)

$$\mathrm{y}(n) = g(f(\mathrm{x}(n))),$$

- nonlinear autoregressive neural network NAR(p)

$$y(n) = f(x(n), x(n-1), ..., x(n-p)).$$

2. For a reactive agent with feedback, the nonlinear input-output model corresponds to the:

- Jordan neural network (JNN)

$$\mathrm{y}(n) = g(f(\mathrm{x}(n), \mathrm{y}(n-1))),$$

- nonlinear autoregressive moving average neural network NARMA(p,q)

$$y(n) = f(x(n), x(n-1), ..., x(n-p), y(n-1), ..., y(n-q)).$$

3. For a reactive agent with an internal state, the nonlinear state space model corresponds to an Elman neural network (ENN) or a simple recurrent neural network (SRN)

$$\mathrm{s}(n) = f(\mathrm{x}(n), \mathrm{s}(n-1)),$$

$$\mathrm{y}(n) = g(\mathrm{s}(n)).$$

4. For a reactive agent with an internal state and feedback, the nonlinear state space model with backward output corresponds to the Jordan-Elman neural network (JENN) proposed by the authors in this article

$$\mathrm{s}(n) = f(\mathrm{x}(n), \mathrm{s}(n-1), \mathrm{y}(n-1)),$$
$$\mathrm{y}(n) = g(\mathrm{s}(n)).$$

5. Jordan-Elman neural network model

Figure 1 shows the structure of the proposed Jordan-Elman Neural Network (JENN) model, which is a recurrent two-layer artificial neural network based on MLP.

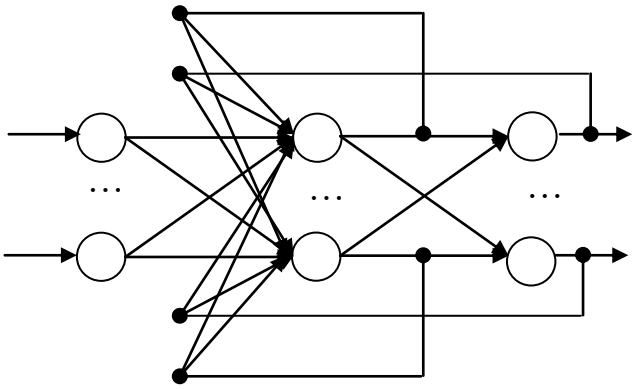

Figure 1: The structure of the Jordan-Elman neural network (JENN) model

1. The hidden layer output signal calculation

$$y_j^{(1)}(n) = f^{(1)}(s_j^{(1)}(n)),$$
$$s_j^{(1)}(n) = w_{0j}^{(1)} + \sum_{i=1}^{N^{(0)}} w_{ij}^{(1)} y_i^{(0)}(n) + \sum_{i=N^{(0)}+1}^{N^{(0)}+N^{(1)}} w_{ij}^{(1)} y_{i-N^{(0)}}^{(1)}(n-1) +$$
$$+ \sum_{i=N^{(0)}+N^{(2)}+1}^{N^{(0)}+N^{(1)}+N^{(2)}} w_{ij}^{(1)} y_{i-N^{(0)}-N^{(1)}}^{(2)}(n-1),\ j \in \overline{1, N^{(1)}},$$

2. The output layer output signal calculation

$$y_j^{(2)}(n) = f^{(2)}(s_j^{(2)}(n)),\ s_j^{(2)}(n) = w_{0j}^{(2)} + \sum_{i=1}^{N^{(1)}} w_{ij}^{(2)} y_i^{(1)}(n),\ j \in \overline{1, N^{(2)}}$$

where $N^{(k)}$ – number of neurons in the k th layer;

$w_{ij}^{(k)}$ – connection weight from the i^{th} neuron to the j^{th} neuron on the k^{th} layer;

$w_{0j}^{(k)}$ – offsets on the k^{th} layer;

$y_j^{(k)}(n)$ – output of the j^{th} neuron on the k^{th} layer at the time n;

$f^{(k)}$ – activation function of neurons of the k^{th} layer (usually $f^{(k)}(s) = sigm(s)$).

6. Performance assessment criterion of the Jordan-Elman neural network model

In this work, to determine the parameters' values of the Jordan-Elman model, the model adequacy criterion was chosen, which means the choice of such values of parameters $W = \{w_{ij}^{(1)},\ w_{ij}^{(2)}\}$, that

deliver a minimum of the mean square error (the difference between the model output and the desired output):

$$F = \frac{1}{P}\sum_{\mu=1}^{P}(y_{\mu}^{(2)} - d_{\mu})^2 \to \min_{W}, \quad (1)$$

where P – test set cardinality.

7. Determining the parameters' values of the Jordan-Elman neural network model based on the Adam method

1. Initialization.

1.1. Number of training iteration $n = 0$.

1.2. Initialization using the uniform distribution on the interval (0,1) or [-0.5, 0.5] weights $w_{ij}^{(1)}(0)$, $i \in \overline{0, N^{(0)} + N^{(1)} + N^{(2)}}$, $j \in \overline{1, N^{(1)}}$, $w_{ij}^{(2)}(0)$, $i \in \overline{0, N^{(1)}}$, $j \in \overline{1, N^{(2)}}$, where $N^{(k)}$ – number of neurons in the k th layer.

1.3. The zero vector of the first moments $\mathbf{m}(-1)$ of length N_q is set $N_q = (N^{(0)} + N^{(1)} + N^{(2)} + 1)N^{(1)} + (N^{(1)} + 1)N^{(2)}$.

1.4. The zero vector of the second moments $\mathbf{v}(-1)$ of length N_q is set $N_q = (N^{(0)} + N^{(1)} + N^{(2)} + 1)N^{(1)} + (N^{(1)} + 1)N^{(2)}$.

1.5. Parameter η is set to determine the learning rate (usually $\eta = 0.001$), the first and second moment decay rates are β_1 and β_2 respectively, $\beta_1, \beta_2 \in [0,1)$ (usually $\beta_1 = 0.9$ and $\beta_2 = 0.999$), and the stability parameter ε to prevent division by zero (usually $\varepsilon = 10^{-8}$).

2. Setting the training set $\{(\mathbf{x}_{\mu}, \mathbf{d}_{\mu}) \mid \mathbf{x}_{\mu} \in R^{N^{(0)}}, \mathbf{d}_{\mu} \in R^{N^{(2)}}\}$, $\mu \in \overline{1, P}$, where $\mathbf{x}_{\mu}$ – μ^{th} learning input vector, $\mathbf{d}_{\mu}$ – μ^{th} learning output vector, $N^{(0)}$ – number of input layer neurons, $N^{(2)}$ – number of output layer neurons, P – the power of the training set. The number of the current pair from the training set $\mu = 1$.

3. The output signal initial calculation for the first layer

$$y_i^{(1)}(n-1) = 0, \; i \in \overline{1, N^{(1)}}.$$

4. The output signal calculation for each layer (forward run)

$$y_i^{(0)}(n) = x_{\mu i},$$

$$y_j^{(1)}(n) = f^{(1)}(s_j^{(1)}(n)),$$

$$s_j^{(1)}(n) = w_{0j}^{(1)} + \sum_{i=1}^{N^{(0)}} w_{ij}^{(1)}(n) y_i^{(0)}(n) + \sum_{i=N^{(0)}+1}^{N^{(0)}+N^{(1)}} w_{ij}^{(1)}(n) y_{i-N^{(0)}}^{(1)}(n-1) +$$

$$+ \sum_{i=N^{(0)}+N^{(2)}+1}^{N^{(0)}+N^{(1)}+N^{(2)}} w_{ij}^{(1)}(n) y_{i-N^{(0)}-N^{(1)}}^{(2)}(n-1), \; j \in \overline{1, N^{(1)}},$$

$$y_j^{(2)}(n) = f^{(2)}(s_j^{(2)}(n)), \; s_j^{(2)}(n) = w_{0j}^{(2)} + \sum_{i=1}^{N^{(1)}} w_{ij}^{(2)}(n) y_i^{(1)}(n), \; j \in \overline{1, N^{(2)}},$$

where $N^{(k)}$ – number of neurons in the k^{th} layer;

$w_{ij}^{(k)}(n)$ – connection weight from the i^{th} neuron to the j^{th} neuron on the k^{th} layer at the time n,

$w_{0j}^{(k)}$ – offsets on the k^{th} layer;

$y_j^{(k)}(n)$ – output of the j^{th} neuron on the k^{th} layer;

$f^{(k)}$ – activation function of neurons of the k^{th} layer.

5. Calculation of ANN error energy

$$E(n)=\frac{1}{2}\sum_{j=1}^{N^{(2)}}e_j^2(n),\ e_j(n)=y_j^{(2)}(n)-d_{\mu j}.$$

6. Setting of synaptic weights based on the generalized delta rule (reverse move)

$$w_{ij}^{(k)}(n+1)=w_{ij}^{(k)}(n)-\eta\frac{\partial E(n)}{\partial w_{ij}^{(k)}(n)},$$

where η – parameter that determines the learning rate (with large η learning is faster, but the risk of getting an incorrect solution increases) $0<\eta<1$,

$$\frac{\partial E(n)}{\partial w_{0j}^{(2)}(n)}=h_j^{(2)}(n),\ i\in\overline{0,N^{(1)}},\ j\in\overline{1,N^{(2)}},$$

$$\frac{\partial E(n)}{\partial w_{ij}^{(2)}(n)}=y_i^{(1)}(n)h_j^{(2)}(n),\ i\in\overline{0,N^{(1)}},\ j\in\overline{1,N^{(2)}},$$

$$\frac{\partial E(n)}{\partial w_{0j}^{(1)}(n)}=h_j^{(1)}(n),\ j\in\overline{1,N^{(1)}},$$

$$\frac{\partial E(n)}{\partial w_{ij}^{(1)}(n)}=\begin{cases}y_i^{(0)}(n)h_j^{(1)}(n), & 0\le i\le N^{(0)}\\ y_{i-N^{(0)}}^{(1)}(n-1)h_j^{(1)}(n), & N^{(0)}<i\le N^{(0)}+N^{(1)}\\ y_{i-N^{(0)}-N^{(1)}}^{(2)}(n-1)h_j^{(1)}(n), & N^{(0)}+N^{(1)}<i\le N^{(0)}+N^{(1)}+N^{(2)}\end{cases},$$

$$i\in\overline{0,N^{(0)}+N^{(1)}+N^{(2)}},\ j\in\overline{1,N^{(1)}},$$

$$h_j^{(k)}(n)=\begin{cases}f'^{(2)}(s_j^{(2)}(n))(y_j^{(2)}(n)-d_{\mu j}), & k=2\\ f'^{(1)}(s_j^{(1)}(n))\sum_{l=1}^{N^{(2)}}w_{jl}^{(2)}(n)h_l^{(2)}(n), & k=1\end{cases}$$

7. The vector of weights is formed

$$\mathbf{w}(n)=\left(w_{01}^{(1)}(n),...,w_{N^{(0)}+N^{(1)}+N^{(2)},N^{(1)}}^{(1)}(n),w_{01}^{(2)}(n),...,w_{N^{(1)}N^{(2)}}^{(2)}(n)\right)^T=$$

$$=\left(w_1(n),...,w_{N_q}(n)\right)^T,\ N_q=(N^{(0)}+N^{(1)}+N^{(2)}+1)N^{(1)}+(N^{(1)}+1)N^{(2)}.$$

8. The vector of partial derivatives (gradient) is formed

$$\mathbf{g}(n)=\left(\frac{\partial E(n)}{\partial w_1(n)},...,\frac{\partial E(n)}{\partial w_{N_w}(n)}\right)^T.$$

9. The vector of the first moments is calculated based on the exponential moving average

$$\mathbf{m}(n)=\beta_1\mathbf{m}(n-1)+(1-\beta_1)\mathbf{g}(n).$$

10. The vector of second moments is calculated based on the exponential moving average

$$\mathbf{v}(n)=\beta_2\mathbf{v}(n-1)+(1-\beta_2)\mathbf{g}^2(n).$$

11. The weight vector is calculated (the first and second moments are corrected due to their initialization by zero and the training step is scaled)

$$\hat{\mathbf{m}}(n)=\mathbf{m}(n)/\left(1-\beta_1^{n+1}\right),$$

$$\hat{\mathbf{v}}(n)=\mathbf{v}(n)/\left(1-\beta_2^{n+1}\right),$$

$$\mathbf{w}(n+1)=\mathbf{w}(n)-\frac{\eta\hat{\mathbf{m}}(n)}{\sqrt{\hat{\mathbf{v}}(n)}+\varepsilon}.$$

12. Checking the termination condition.

If $n \bmod P>0$, then $\mu=\mu+1$, $n=n+1$, go to 4.

If $n \bmod P = 0$ and $\frac{1}{P}\sum_{s=1}^{P} E(n-P+s) > \varepsilon$, then $n = n+1$, go to 2.

If $n \bmod P = 0$ and $\frac{1}{P}\sum_{s=1}^{P} E(n-P+s) < \varepsilon$, then end.

8. Determining the parameters' values of the Jordan-Elman neural network model based on the modified metaheuristic method for a charged system search

Charged system search (CSS) was proposed by Kaveh and Talatahari and is based on Coulomb's law from electrostatics and Newton's second law from mechanics. The position of each particle in space corresponds to a solution (vector of parameters' values). The target function is the adequacy criterion of the neural network model (1). In this paper, a modification of this method is made - annealing simulation is introduced, which allows you to explore the entire search space at the initial iterations (the exploitation parameter is small, the exploration parameter is large), and at the final iterations the search becomes directed (the exploitation parameter is large, the exploration parameter is small), while the operation and research parameters change non-linearly.

1. Initialization.

1.1. Setting the probability of generating a position randomly P^{gen}; probability of modifying the position selected from memory P^{update}, parameter δ for generating a new position, moreover $0 < \delta < 1$, initial temperature T_0, cooling coefficient β.

1.2. Setting the maximum number of iterations N, population size K, memory size $L_{\max}$, particle position vector length M (number of neural network model parameters), minimum and maximum values for the position vector $x_j^{\min}, x_j^{\max}$, $j \in \overline{1,M}$, minimum and maximum values for the velocity vector $v_j^{\min}, v_j^{\max}$, $j \in \overline{1,M}$.

1.3. Randomly generating the best position vector

$$x^* = (x_1^*,...,x_M^*),\ x_j^* = x_j^{\min} + (x_j^{\max} - x_j^{\min})U(0,1),$$

where $U(0,1)$ – is a function that returns a uniformly distributed random number in the range $[0,1]$.

1.4. Creation of the initial population P.

1.4.1. Particle number $k = 1$, $P = \varnothing$.

1.4.2. Randomly generating a position vector x_k

$$x_k = (x_{k1},...,x_{kM}),\ x_{kj} = x_j^{\min} + (x_j^{\max} - x_j^{\min})U(0,1).$$

1.4.3. Randomly generating a velocity vector v_k

$$v_k = (v_{k1},...,v_{kM}),\ v_{ij} = v_j^{\min} + (v_j^{\max} - v_j^{\min})U(0,1).$$

1.4.4. If $(x_k, v_k) \notin P$, then $P = P \cup \{(x_k, v_k)\}$, increase particle number k by one.

1.4.5. If $k \le K$, then go to 1.4.2.

1.5. Order P by target function, i.e., $F(x_k) < F(x_{k+1})$.

1.6. Put $L_{\max}$ best (first) particles into the memory Q.

2. Iteration number $n = 1$.

3. Determine the best particle in terms of the target function

$$k^* = \arg\min_k F(x_k),\ k \in \overline{1,K}.$$

4. If $F(x_{k^*}) < F(x^*)$, then $x^* = x_{k^*}$.

5. The sphere radius calculation

$$a = 0.1 \cdot \max_j \{x_j^{\max} - x_j^{\min}\},\ j \in \overline{1,M}.$$

6. The particle charge calculation

$$q_k = \frac{F(x_k) - \max\limits_s F(x_s)}{\min\limits_s F(x_s) - \max\limits_s F(x_s)},\ k \in \overline{1,K}.$$

7. Calculation of the gap between two charged particles

$$r_{kl} = \frac{\| x_k - x_l \|}{\| (x_k + x_l)/2 - x_{k^*} \|},\ k,l \in \overline{1,K},$$

where $\|\cdot\|$ – is the norm (for example, Euclid).

8. Determine if one particle is moving towards another, and it is believed that all good particles can attract bad ones, but only some bad particles can attract good ones

$$p_{kl} = \begin{cases} 1, & \dfrac{F(x_k) - F(x_{k^*})}{F(x_l) - F(x_k)} > U(0,1) \vee F(x_l) > F(x_k) \\ 0, & other \end{cases},\ k,l \in \overline{1,K}.$$

9. Determine if the particle is inside the sphere

$$s_{kl} = \begin{cases} 1, & r_{kl} < a \\ 0, & r_{kl} \geq a \end{cases},\ k,l \in \overline{1,K}.$$

10. Calculate the resulting electrical force acting on particles inside or outside the sphere, taking into account Coulomb's law

$$f_l = q_l \sum_{k,k \neq l} \left(\frac{q_k}{a^3} r_{kl} s_{kl} + \frac{q_k}{r_{kl}^2} (1 - s_{kl}) \right) p_{kl} (x_k - x_l),\ l \in \overline{1,K}.$$

11. Calculate particle acceleration

$$a_k = \frac{f_k}{m_k},\ k \in \overline{1,K}.$$

12. Modify the position of the particles, taking into account Newton's second law and simulated annealing.

12.1. $x_k^{old} = x_k,\ k \in \overline{1,K}$.

12.2. $\lambda_1 = U(0,1)$.

12.3. $\lambda_2 = U(0,1)$.

12.4. $\alpha_1(n) = (1 - \exp(-1/T(n)))$, $T(n) = \beta^n T_0$.

12.5. $\alpha_2(n) = \exp(-1/T(n))$, $T(n) = \beta^n T_0$.

12.6. $x_k = x_k^{old} + \lambda_1 \alpha_1(n) a_k \Delta t^2 + \lambda_2 \alpha_2(n) v_k \Delta t,\ k \in \overline{1,K}$,

where $\Delta t = 1$ – time quantization step,

m_k – the mass of the l th particle, coinciding with the value of its charge, i.e., $m_k = q_k$,

$T(n)$ – annealing temperature at iteration n,

T_0 – initial annealing temperature,

β – cooling factor,

$\alpha_1(n)$ – operation parameter at iteration n,

$\alpha_2(n)$ – research parameter at iteration n.

13. Modify the position of particles that are out of bounds.

13.1. Particle number $k = 1$.

13.2. If $x_{kj} \in [x_j^{\min}, x_j^{\max}]$, then go to 13.9.

13.3. If $U(0,1) < P^{gen}$, then go to 13.8.

13.4. The m^{th} particle is randomly selected from the memory, i.e.

$$m = round(1 + (L_{\max} - 1)U(0,1)),$$

where $round()$ – is a function that rounds a number to the nearest integer.

13.5. If $U(0,1) > P^{update}$, then $x_k = \widetilde{x}_m$, go to 13.9.

13.6. Generation of solution x_k from solution $\widetilde{x}_m$.

13.6.1. $x_{kj} = \widetilde{x}_{mj} + \delta(x_j^{\max} - x_j^{\min})(-1 + 2U(0,1))$.

13.6.2. $x_{kj} = \max\{x_j^{\min}, x_{kj}\}$, $x_{kj} = \min\{x_j^{\max}, x_{kj}\}$.

13.7. Go to 13.9.

13.8. Random generation of position x_k

$$x_{kj} = x_j^{\min} + (x_j^{\max} - x_j^{\min})U(0,1),\ j \in \overline{1,M}.$$

13.9. If $k < K$, then increase the number of particles k by one and go to 13.2.

14. Modify particle speed

$$v_k = \frac{x_k - x_k^{old}}{\Delta t},\ k \in \overline{1,K}.$$

15. Order P by target function, i.e., $F(x_k) < F(x_{k+1})$.

16. Modify memory.

Merge particles from the memory Q and population P, order pool $P \cup H$ by target function, i.e., $F(x_k) < F(x_{k+1})$, and put the top $L_{\max}$ best (first) particles from the pool $P \cup H$ into the memory Q.

17. Stop condition.

If $n < N$, then increase the iteration number n by one and go to 3.

The result is x^*.

9. Experiments and results

The simulation of the determination process of the parameters' values of the neural network model based on the modified method of charged system search (CSS) was carried out in the Matlab package using the Parallel Computing Toolbox. It is proposed to perform parallel processing of particles using the parfor parallel loop, which is included in Parallel Computing Toolbox, since the formation of each particle and its velocity in step 1, the modification of the acceleration of each particle, the position of each particle, the velocity of each particle, in steps 6-11, 12-13 and 14, respectively, occurs independently of other particles, and the order of formation and modification of particles is arbitrary.

The parallel loop parfor replaces the sequential for loop and is based on OpenMP technology, but unlike it, it can be used not only on a local multi-core machine but also on a cluster. The advantage of this approach over CUDA and MPI technologies [23, 24] (represented by the spmd block in Parallel Computing Toolbox) is the simplicity and clarity of technical implementation. Due to the small number of particles, it becomes possible to perform the formation and modification of each particle on the corresponding physical core of the machines' processors united in a cluster.

The size of the swarm of particles K=40, the number of iterations N=100, the memory size $L_{\max}$=K/4, the probability of randomly generating a position P^{gen}=0.05, the probability of modifying a position selected from memory P^{update}=0.1, the parameter for generating a new position δ=0.1, initial temperature $T_0 = 106$, cooling factor $\beta = 0.94$ were selected.

The function of decreasing the annealing temperature is determined by the formula $T(n) = \beta^n T_0$ and is shown in Figure 2.

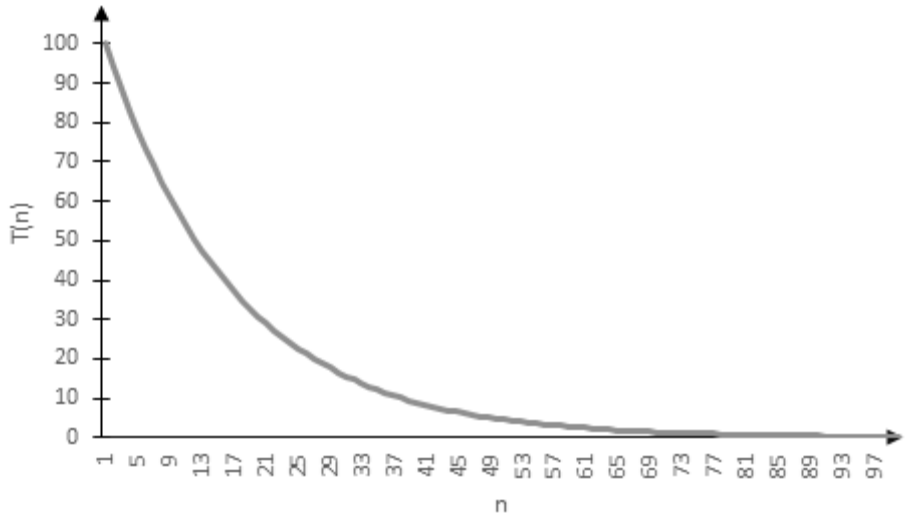

Figure 2: The function of decreasing the annealing temperature

The dependence (Figure 2) of the annealing temperature on the iteration number shows that the annealing temperature decreases with an increase in the iteration number.

The operating parameter is determined by the formula $\alpha_1(n) = \exp(-1/T(n))$ and is shown in Figure 3.

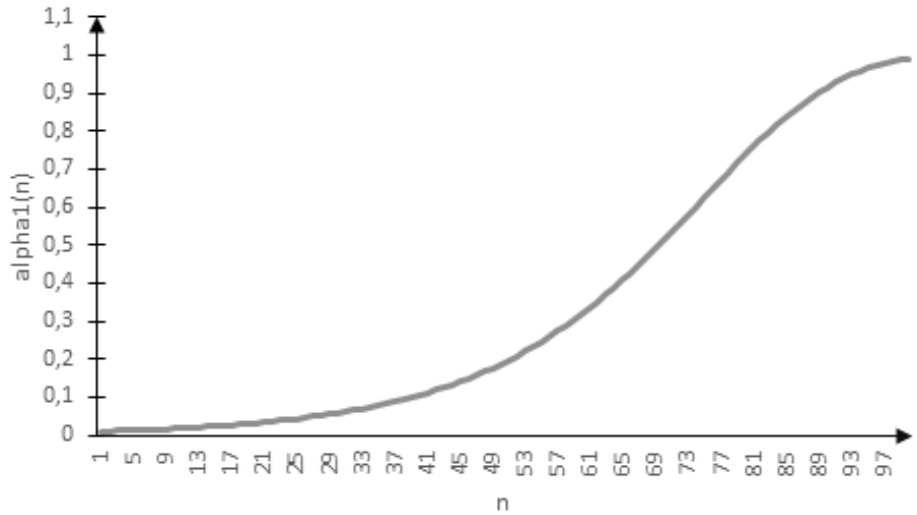

Figure 3: Operating parameter

The dependence (Figure 3) of the operating parameter on the iteration number shows that the value of this parameter increases non-linearly with time.

The research parameter is determined by the formula $\alpha_2(n) = \exp(-1/T(n))$ and is shown in Figure 4.

The dependence (Figure 4) of the research parameter on the iteration number shows that the value of this parameter decreases non-linearly over time.

To determine the structure of the Jordan-Elman neural network model, i.e., to determine the number of hidden neurons, several experiments were carried out, the results of which are shown in Figure 5. As input data for determining the values of the parameters of the Jordan-Elman neural network model, a selection of values based on the data of the logistics company «Ekol Ukraine» was used. The number of input neurons was 8. The criterion for choosing the structure of the neural network model was the minimum mean square prediction error (MSE).

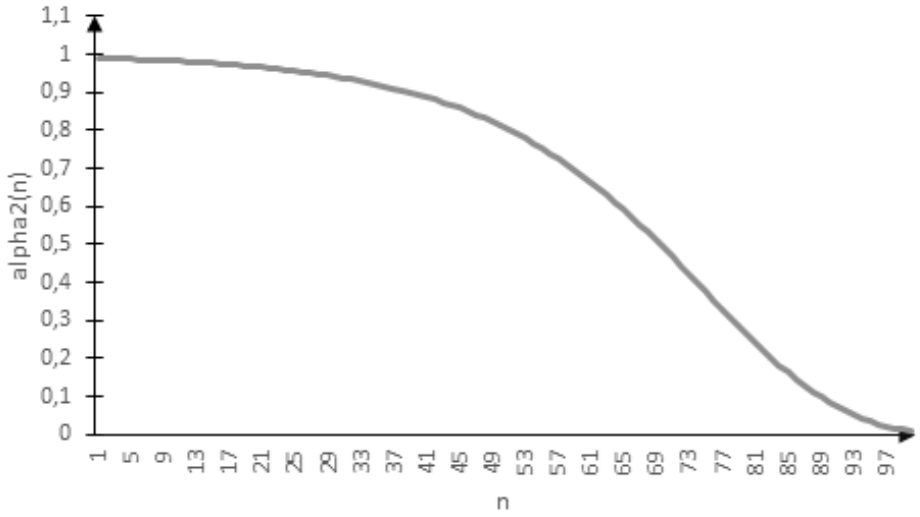

Figure 4: Research parameter

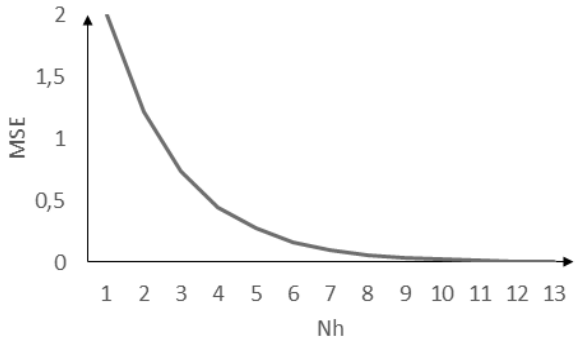

Figure 5: Graph of the dependence of the MSE value on the number of hidden neurons

As can be seen from the graph in Figure 5, with an increase in the number of hidden neurons, the error value decreases. For prediction, it is sufficient to use 8 neurons of the hidden layer, since with a further increase in the number of neurons in the hidden layer, the change in the error value is insignificant. Thus, the number of hidden neurons coincided with the number of input neurons.

Methods for determining the values of the parameters of the Jordan-Elman neural network model by the criterion of minimum mean square error (MSE) of the forecast and computational complexity were investigated in this work (Table 1), where $N^{(k)}$ is the number of neurons in the k^{th} layer, P is the power of the training set, N is the number of iterations, K is the particle swarm size.

Table 1
Comparative characteristics of methods for determining parameter values

Criterion \ Method	Adam	modified CSS
Minimum MSE of the forecast	0.9	0.95
Computational complexity	$\approx N^{(0)} * N^{(1)} * N^{(2)} * N^{(1)} * P * N$	$\approx N^{(0)} * N^{(1)} * N^{(2)} * N^{(1)} * P * N * K$ (without parallelism) $\approx N^{(0)} * N^{(1)} * N^{(2)} * N^{(1)} * P * N$ (with parallelism)

According to Table 1, in terms of MSE prediction, the modified CSS method gives the best results, in terms of computational complexity without using parallelism, the Adam method gives the best results, and in the case of parallelism, these methods give the same results.

10. Conclusions

The article examines the problem of increasing the efficiency of computer agents in supply chains. To solve this problem, the existing computer agents of multi-agent systems were investigated. These studies have shown that today the most effective is the use of a computer agent functioning model based on the connectionist approach.

To expand the range of tasks solved by agents, the article proposes a reactive agent with feedback, which makes a decision based on perception or a sequence of perceptions and a previous action or a sequence of previous actions. Also proposed is a reactive agent with an internal state and feedback, which is an extension of the reactive agent with an internal state and makes a decision based on perception, previous internal state and previous action.

For a reactive agent with internal state and feedback, a Jordan-Elman artificial neural network was proposed, which is a combination of Jordan and Elman neural networks. In the course of a numerical study, the structure of its model was determined. The experiments performed showed that when the number of hidden neurons is not less than the number of neurons in the input layer, the value of the mean square error does not change significantly, and the selected network gives results with a minimum error.

Methods for determining the parameters' values of the proposed Jordan-Elman neural network model were proposed. This made it possible to ensure high speed and accuracy of calculations based on the model. The proposed metaheuristic method for determining parameter values allows for parallelization and uses annealing simulation, which allows the entire search space to be explored in the initial iterations, and in the final iterations the search becomes directed.

The proposed methods for determining the parameters' values are intended for software implementation in the Matlab package using the Parallel Computing Toolbox, which speeds up the process of determining the parameters' values of the Jordan-Elman neural network model.

The developed neural network model and methods for determining its parameters make it possible to increase the efficiency of the agent's functioning. The software that implements the proposed method for creating an artificial neural networks-based computer agent was developed and studied with the database of the «Ekol Ukraine» logistics company.

The experiments performed have confirmed the efficiency of the developed software and allow us to recommend it for practical use in solving supply chain management problems using multi-agent systems. The prospects for further research are to test the proposed method on a wider set of test databases.

11. References

[1] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative competitive environments, in: Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17), 2017, pp. 6382–6393.

[2] C. M. Macal, Everything you need to know about agent-based modelling and simulation, Journal of Simulation. 10.2 (2016) 144-156. doi: 10.1057/jos.2016.7.

[3] S. Russell, P. Norvig, Artificial Intelligence: a Modem Approach, Englewood Cliffs, NJ: Prentice Hall PTR, 2020.

[4] J. Rocha, Multi-agent Systems, IntechOpen, London, 2017.

[5] K.-L. Du, K. M. S. Swamy, Neural Networks and Statistical Learning, Springer-Verlag, London, 2014.

[6] A. Wysocki, M. Ławryńczuk, Predictive control of a multivariable neutralisation process using Elman neural networks, in: Advances in Intelligent Systems and Computing, Springer: Heidelberg, 2015, pp. 335-344. doi: 10.1007/978-3-319-15796-234.

[7] A. Wysocki, M. Ławryńczuk, Jordan neural network for modelling and predictive control of dynamic systems, in: Proceedings of the 20th International Conference on Methods and Models in Automation and Robotics (MMAR), 2015, pp. 145-150. doi: 10.1109/MMAR.2015.7283862.
[8] R. Dey, F. M. Salem, Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks, arXiv:1701.05923, 2017. – URL: https://arxiv.org/ftp/arxiv/papers/1701/1701.05923.pdf.
[9] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder-decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1724–1734. doi: 10.3115/v1/D14-1179.
[10] A. H. S. Hamdany, R. R. O. Al-Nima, L. H. Albak, Translating cuneiform symbols using artificial neural network, in: TELKOMNIKA Telecommunication, Computing, Electronics and Control, volume 19, No. 2, 2021, pp. 438-443. doi: 10.12928/telkomnika.v19i2.16134.
[11] M. Sundermeyer, T. Alkhouli, J. Wuebker, H. Ney, Translation modeling with bidirectional recurrent neural networks, in: Proceedings of the Conference on Empirical Methods on Natural Language Processing, 2014, pp. 14-25.
[12] M. Berglund, T. Raiko, M. Honkala, L. Kärkkäinen, A. Vetek, J. Karhunen, Bidirectional recurrent neural networks as generative models, in: Proceedings of the 28th International Conference on Neural Information Processing Systems, 2015, pp. 856−864.
[13] P. Potash, A. Romanov, A. Rumshisky, Ghostwriter: using an LSTM for automatic rap lyric generation, in: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, 2015, pp. 1919– 1924. doi:10.18653/v1/D15-1221.
[14] E. Kiperwasser, Y. Goldberg, Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations, Transactions of the Association for Computational Linguistics 4 (2016) 313–327. doi: 10.1162/tacl_a_00101.
[15] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint arXiv:1412.3555, 2014.
[16] S. A. Khan, S. M. D. Khalid, M. A. Shahzad, F. Shafait, Table structure extraction with bi-directional gated recurrent unit networks, in: International Conference on Document Analysis and Recognition (ICDAR), volume 4, No. 2, 2019, pp. 78–88. doi: 10.1109/ICDAR.2019.00220.
[17] A. Fischer, C. Igel, Training Restricted Boltzmann Machines: An Introduction, Pattern Recognition 47 (2014) 25-39. doi: 10.1016/j.patcog.2013.05.025.
[18] E. Fedorov, V. Lukashenko, V. Patrushev, A. Lukashenko, K. Rudakov, S. Mitsenko, The method of intelligent image processing based on a three-channel purely convolutional neural network, in: CEUR Workshop Proceedings, volume 2255, 2018, pp. 336–351. doi: 10.1109/EWDTS.2013.6673185.
[19] I. Loshchilov, CMA-ES with restarts for solving CEC 2013 benchmark problems, in: Proceedings of IEEE congress on evolutionary computation (CEC 2013), 2013, pp. 369–376. doi: 10.1109/CEC.2013.6557593.
[20] M. R. H. M. Rasdi, I. Musirin, Z. A. Hamid, H. C. M. Haris, Gravitational search algorithm application in optimal allocation and sizing of multi distributed generation, in: Proceedings of 8th International Power Engineering and Optimization Conference, Langkawi, Malaysia, 2014, pp. 364–368.
[21] J. Radosavljevic, M. Jevtic, D. Klimenta, Energy and operation management of a microgrid using particle swarm optimization, Engineering Optimization 48(5) (2016) 811–830. doi.org/10.1080/0305215X.2015.1057135
[22] M. Petrović, J. Petronijević, M. Mitić, N. Vuković, Z. Miljković, B. Babić, The Antlion optimization algorithm for integrated process planning and scheduling, Applied Mechanics and Materials, 834(1) (2016) 187–192. doi: 10.4028/www.scientific.net/AMM.834.187
[23] G. G. Shvachych, O. V. Ivaschenko, V. V. Busygin, Ye. Ye. Fedorov, Parallel computational algorithms in thermal processes in metallurgy and mining, Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu, 4 (2018) 129–137. doi: 10.29202/nvngu/2018-4/19.
[24] G. Shlomchak, G. Shvachych, B. Moroz, E. Fedorov, D. Kozenkov, Automated control of temperature regimes of alloyed steel products based on multiprocessors computing systems, Metalurgija, 58 (2019) 299-302.