Convergence and integration of artificial neural networks with knowledge bases in next-generation intelligent computer systems

Mikhail Kovalev
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
michail.kovalev7@gmail.com

Aliaksandr Kroshchanka
Brest State Technical University
Brest, Belarus
kroschenko@gmail.com

Vladimir Golovko
Brest State Technical University
Brest, Belarus
vladimir.golovko@gmail.com

***Abstract*—In the article, an approach to the integration and convergence of artificial neural networks with knowledge bases in next-generation intelligent computer systems through the representation and interpretation of artificial neural networks in a knowledge base is considered. The syntax, denotational, and operational semantics of the language for representing neural network methods in knowledge bases are described. The stages of building of neural network problem-solving methods with the help of intelligent framework for designing artificial neural networks are described.**

***Keywords*—problem-solving model, ontological approach, neuro-symbolic AI, artificial neural network**

I. Introduction

The term of a next-generation intelligent computer system implies that such systems, among others, have the following capabilities [1]:

- the ability to constantly improve the quality of problem solving;
- the ability to acquire skills for solving fundamentally new problems;
- the ability to explain their own decisions;
- the ability to find and eliminate errors in their own decisions (the ability to introspect).

Ensuring the above abilities is fundamentally possible in the concept proposed by the OSTIS project [1] due to the unification of the representation and ontological structuring of knowledge describing *problems*, subject domains within which problems are solved, and *problem-solving methods*.

Representation of various problems-solving methods in a common knowledge base ensures the semantic compatibility of these methods. When solving a problem using such methods, the system does not interact with them on the principle of "inputs–outputs". On the contrary, a common memory allows real-time transformation of input knowledge using any available methods, which provides the ability to introspect and explain the decisions of the system.

Promising and actively developing problem-solving methods are artificial neural networks (ANN), which is determined, on the one hand, by the evolution of the theory of ANN and, on the other hand, by the hardware capabilities of the machines that are used to train them.

The advantages of ANN include the ability to solve problems with unknown patterns, as well as the ability to solve problems without the need to develop problem-oriented approaches.

However, most neural network models work like a "black box" [2], which is one of the main disadvantages of this problem-solving method. Modern problems increasingly require explanation of their solution. A whole direction of Explainable AI has appeared, within which various attempts are made to explain the decisions of ANN [3], [4]. Approaches that propose the integration of neural networks with knowledge bases are being developed [5]–[7].

As a disadvantage of ANN we can also name the heuristic nature of the process of finding optimal architectures of models and the parameters for their training, as well as the high requirements for the scope of knowledge of neural network models researchers.

Based on the above abilities, the presence of which must be ensured in next-generation intelligent computer systems, the problem of developing an approach to the integration of ANN into the knowledge base of an intelligent system arises, both as a problem-solving method and as an object of automatic design of new methods. The solution of this problem will allow overcoming the above disadvantages of the neural network method.

The purpose of the research is to expand the range of problems solved by intelligent systems by developing a set of models, methods, and tools for representing, designing, and processing artificial neural networks in intelligent systems and integrating them with other problem-solving models.

II. Proposed approach

The basis of the proposed approach is the usage of the OSTIS technology and its basic principles [8]. Intelligent systems developed using the OSTIS technology are called ostis-systems. Any ostis-system consists of a knowledge base, a problem solver, and a user interface.

The problem solver performes the processing of fragments of the knowledge base. At the operational level, processing means adding, searching, editing, and deleting sc-nodes and sc-connectors of the knowledge base. On the semantic level, such an operation is an *action performed in the memory of an action subject*, where, in the general case, the subject is an ostis-system and the knowledge base is its memory. An *action* is defined as the influence of one entity (or some set of entities) to another entity (or some set of other entities) according to some purpose.

Actions are performed according to the set problems. A *problem* is a formal specification of some action, sufficient to perform this action by some subject. Depending on a particular class of problems, it is possible to describe both the internal state of the intelligent system itself and the required state of the external environment [9].

Similar problems are grouped into classes, for which generalized problem formulations are described. The following classes of problems for ANN are defined [10]:

- *The classification problem*. The problem of constructing a classifier, i.e. a mapping $\tilde{c} : X \to C$, where $X \in \mathbb{R}^m$ is the feature space of the input example, $C = C_1, C_2, ...C_k$ is a finite and usually small set of class labels.
- *The regression problem*. The problem of constructing an evaluation function by examples $(x_i, f(x_i))$, where $f(x)$ is an unknown function. The *evaluation function* is a mapping of the form $\tilde{f} : X \to \mathbb{R}$, where $X \in \mathbb{R}^m$ is the feature space of e.a.p.
- *The clustering problem*. The problem of constructing a function $a : X \to Y$ that matches any object $x \in X$ with a cluster number $y \in Y$ with a certain distance metric $\rho(x, x')$, where *X* is a set of objects, *Y* is a set of cluster numbers (names, labels), $x, x' \in X$.
- *The problem of decreasing the dimensionality of the feature space*. The problem of constructing a function $h : X \to Y$ that preserves the given relations between points of sets X and Y, where $X \subset \mathbb{R}^p$, $Y = h(X) \subset \mathbb{R}^q$, $q < p$.
- *The control problem*. The problem of constructing a model-regulator for the state of a complex dynamic object.
- *The filtering problem*. The problem of building a model that cleans the original signal containing some noise and reduces the influence of random errors in the signal.
- *The detection problem*. It is a special case of the classification and regression problems. The problem of constructing a model that performs the detection of objects of certain types in photo and video images.
- *The problem with associative memory*. The problem of constructing a model that allows reconstructing the original example based on previously saved examples.

For classes of problems, classes of methods for their solution are formulated. A *problem-solving method* is defined as a problem-solving program of the corresponding class, which can be either procedural or declarative. In turn, a *class of problem-solving methods* is defined as a set of all possible problem-solving methods having a common language for representing these methods. The method representation language allows describing the syntactic, denotational, and operational semantics of this method.

In this article, we propose to consider ANN as a class of problem-solving methods with its own representation language. According to the OSTIS technology, the specification of a class of problem-solving methods is reduced to the specification of the corresponding method representation language, i.e. to the description of its syntactic, denotational, and operational semantics.

To achieve semantic compatibility with other problem-solving methods of the OSTIS technology, it is proposed to describe neural network methods within semantic memory, accordingly, the syntax of the representation language of neural network problem-solving methods is the syntax of the SC-code used in the OSTIS technology for knowledge representation.

Thus, in order to add neural network problem-solving methods to the stack of the OSTIS technology and thus expand the range of problems solved by ostis-systems, it is necessary to describe the denotational and operational semantics of the representation language for the neural network problem-solving method.

The denotational semantics of neural network method representation language is described within the subject domain and its corresponding ontology of a neural network method. This model is described in detail in **Section III**.

The operational semantics of any problem-solving method representation language is the specification of a family of agents providing the interpretation of any method belonging to the corresponding method class. This family is an interpreter of the corresponding problem-solving method. Within the OSTIS technology, such an interpreter is called a *problem-solving model*. Since the OSTIS technology uses a multi-agent approach, the development of a neural network problem-solving model is reduced to the development of an agent-oriented model of ANN interpretation. This model is described in **Section IV**.

A *skill* is a method, the interpretation of which can be fully carried out by a given cybernetic system, in

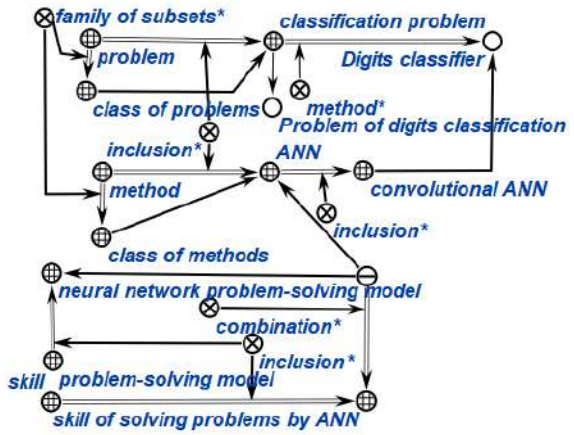

Figure 1. A fragment of the set-theoretic ontology of ANN

the memory of which the specified method is stored [9]. Thus, forming the specification for the neural network problem-solving method and neural network problem-solving model in the ostis-system, we can say that such system possesses the skill of problem solving with the help of ANN.

In Figure 1, a fragment of the ANN ontology is shown, describing the relation of such concepts and nodes as:

- a class of problems that can be solved by ANN (for example, the class of classification problems);
- a class of neural network problem-solving methods;
- a neural network problem-solving model;
- a skill in problem solving with the help of ANN;
- specific problems and methods of their solution (for example, a specific trained convolutional ANN).

The usage of ANN as a problem-solving method implies the usage of an already designed and trained ANN. However, the presence of a neural network method description language in ostis-system memory opens the way for automation of the design and training ANN processes themselves. Such automation is represented by separate classes of problems and the corresponding skills for their solution. The approach to such automation is described in **Section V**.

III. Denotational semantics of the neural network representation language

As it was already mentioned, the denotational semantics of neural network method representation language is described within the subject domain (SD) and its corresponding neural network method ontology. The SD of neural network methods is a private SD of the method.

The maximum class of artificial neural network research objects is an *artificial neural network*.

The SD of a neural network method and key elements of its ontology are described in [10]. In this article, an extension of the SD of neural network methods, described in [10], is represented.

Let us demonstrate an updated classification of neural network methods (the added classes are in bold):

artificial neural network
:= [neural network method]
⇐ *inclusion**:
method
⇒ *subdividing**:
Typology of ANN on the basis of the directivity of connections^
= {
- *ANN with direct connections*
  ⇒ *decomposition**:
  {• *perceptron*
    ⇒ *decomposition**:
    {• *Rosenblatt perceptron*
    • *autoencoder ANN*
    }
  • *support vector machine*
  • *ANN of radial basis functions*
  • ***convolutional ANN***
  }
- *ANN with inverse connections*
  ⇒ *decomposition**:
  {• *Hopfield ANN*
  • *Hamming ANN*
  }
- *recurrent ANN*
  ⇒ *decomposition**:
  {• *Jordan ANN*
  • *Elman ANN*
  • *multi-recurrent ANN*
  • *LSTM-element*
  • *GRU-element*
  }

}
⇒ *subdividing**:
Typology of ANN on the basis of completeness of connections^
= {• *fully connected ANN*
• *weakly connected ANN*
}

The concepts for describing metrics of neural network methods effectiveness are also added in the SD of neural network methods. These metrics are taken into account by the problem solver when deciding to use one or another neural network method.

Metrics can be classified according to the type of problem to be solved.

ANN quality assessment metric

⇒ *subdividing**:
*Metric typology by problems^*
= {• *classification metrics*
⇒ *decomposition**:
{• *ANN precision*
• *ANN completeness*
• *F1-metric*
}
• *regression metrics*
⇒ *decomposition**:
{• *MAE*
• *MAPE*
• *RMSE*
}
}

ANN precision

:= [precision]
:= [proportion of correctly identified positive outcomes in the total number of outcomes that were identified as positive]
⇒ *formula**:
[

$$PRE = \frac{TP}{TP + FP}$$

where *TP* and *FP* are the number of true-positive and false-positive predictions of the neural network, respectively]

ANN completeness

:= [recall]
:= [proportion of correctly identified positive outcomes in the total number of positive outcomes]
⇒ *formula**:
[

$$REC = \frac{TP}{TP + FN}$$

where *TP* and *FN* are the number of true-positive and false-negative predictions of the neural network, respectively]

F1-metric

⇒ *formula**:
[

$$F1 = 2 * \frac{PRE * REC}{PRE + REC}$$

where *PRE* and *REC* are the accuracy and completeness of ANN, respectively]

MAE

:= [mean absolute error]
⇒ *formula**:
[$\frac{1}{N}\sum_{i=1}^{N}|y^i_{etalon} - y^i_{predicted}|$, y^i_{etalon} – the reference value, $y^i_{predicted}$ – the value obtained by the ANN, *N* – the size of the training dataset]

MAPE

:= [mean absolute percentage error]
⇒ *formula**:
[$\frac{1}{N}\sum_{i=1}^{N}\frac{y^i_{etalon} - y^i_{predicted}|}{y^i_{etalon}} * 100\%$,
y^i_{etalon} – the reference value,
$y^i_{predicted}$ – the value obtained by the ANN,
N – the size of the training dataset]

RMSE

:= [root mean squared error]
⇒ *formula**:
[$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y^i_{etalon} - y^i_{predicted})^2}$, y^i_{etalon} – the reference value, $y^i_{predicted}$ – the value obtained by the ANN,
N – the size of the training dataset]

IV. Operational semantics of the neural network representation language

Operational semantics of neural network representation language is defined by the agent-oriented model of artificial neural network interpretation and specification of corresponding actions.

A neural network method is described in the form of a program in some programming language, which can be either external in relation to the ostis-system or internal (at the moment, an SCP language). Each such programming language corresponds to some private subject domain of the SD of neural network methods.

Subject domain of neural network methods

:= [Subject domain of artificial neural networks]
⇒ *private subject domain**:
{• *Subject domain of neural network methods in SCP*
• *Subject domain of neural network methods in Python*
• *Subject domain of neural network methods in C++*
}

In the case of description of a neural network method in an external language, such method is described in the corresponding subject domain, within which the action for interpretation of this method is also specified. This action corresponds to an agent implemented in the corresponding programming language.

However, to achieve convergence and integration, it is necessary to describe neural network methods in the internal language of the ostis-system, which is SCP [1].

An scp-program is a sequence of generalized specifications (templates) of scp-operators. Each scp-operator is an action in ostis-system memory (sc-memory). During interpreting an scp-program, the abstract sc-agent of creating scp-processes creates an scp-process, taking into account the specific scp-program interpretation parameters.

In many cases that means substituting arguments in the generalized scp-operator specifications of the program and generating specific instances of these programs (methods). Then, the initial operator is added to the set of real entities, and the program execution begins.

Thus, the interpretation of an scp-program comes down to agent-based processing of actions in the scp-memory, which are scp-operators.

The neural network method representation language in SCP is an extension of the SCP language. It is extended at the expense of actions, specific to the SD of ANN. The subject domain and its corresponding ontology of neural network methods in SCP describes the specification of actions for interpretation of ANN within ostis-system memory, which extend the range of standard scp-operators. The following hierarchy of such actions can be distinguished:

action for interpreting the ANN layer
⇒ *decomposition**:
{• *action for calculating the weighted sum of all neurons of the layer*
• *action for calculating the activation function for all neurons of the layer*
• *action for interpreting the convolutional layer*
• *action for interpreting the pooling layer*
}

To describe the specification of the above actions, it is necessary to introduce the concepts of *oriented number set* and *matrix* using which the input values of ANN, output values of ANN, weight matrices, and so on are specified.

Each element of an oriented number set is some number. The numbers can be represented as sc-nodes or with a string representation of the whole set, for which a special relation *string representation of the oriented number set** is used. This relation was introduced in order to optimize some implementation options of the agent interpreting action using the concept of oriented number set.

oriented number set
:= [oriented number set]
⇐ *inclusion**:
number
⇐ *inclusion**:
oriented set
⇐ *first domain**:
*string representation of an oriented number set**

A *matrix* is an oriented set of oriented sets of equal power numbers.

1. Action for calculating the weighted sum of all neurons of the layer

The (*objects'*) arguments of this action are set by the following relations:

input vector'
⇒ *first domain**:
action for interpreting the ANN layer
⇒ *second domain**:
oriented number set

matrix of neuron synapse weights of the layer'
⇒ *first domain**:
action for processing ANN
⇒ *second domain**:
matrix

The result of the (*result'*) action is an oriented number set, which is the weighted sum of neurons of the corresponding layer.

An example for the specification of the action for calculating the weighted sum of all neurons of the layer for a layer with two neurons and an input vector of dimension 2 is shown in Figure 2.

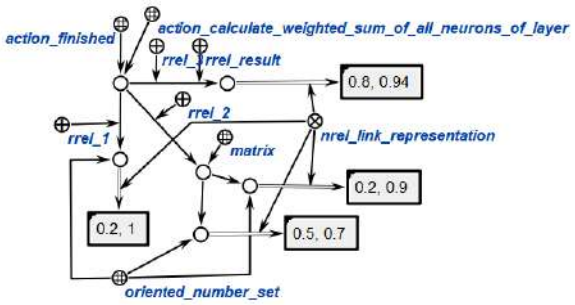

Figure 2. An example for the specification of the action for calculating the weighted sum of all neurons of the layer

2. Action for calculating the activation function for all neurons of the layer

The arguments of this action are set by the following relations:

vector of weighted sums of layer neurons'
⇒ *first domain**:
action for processing ANN'
⇒ *second domain**:
oriented number set

threshold vector of layer neurons'
⇒ *first domain**:
action for processing ANN'
⇒ *second domain**:
oriented number set

activation function'
⇒ *first domain**:
action for processing ANN'
⇒ *second domain**:
function

The result of the action is an oriented number set, which are the output values of the layer neurons.

3. Action for interpreting the convolutional layer

The arguments of this action are set by the following relations:

input matrix'
⇒ *first domain**:
action for processing ANN
⇒ *second domain**:
matrix

convolution kernel'
⇒ *first domain**:
action for interpreting the convolutional layer
⇒ *second domain**:
matrix

convolution step'
⇒ *first domain**:
action for interpreting the convolutional layer
⇒ *second domain**:
number

The result of the action is the matrix resulting from the convolution of the input matrix with the convolution kernel.

4. Action for interpreting the pooling layer

The arguments of this action are defined by the following relations:

input matrix'
⇒ *first domain**:
action for processing ANN
⇒ *second domain**:
matrix

pooling window size'
⇒ *first domain**:
action for interpreting the pooling layer
⇒ *second domain**:
matrix

pooling window step'
⇒ *first domain**:
action for interpreting the pooling layer
⇒ *second domain**:
number

The result of the action is the matrix obtained as a result of pooling the input matrix.

If it is necessary to specify different arguments for neurons of the same layer, it is possible to specify the corresponding actions, however, this was not used in this work due to the poor knowledge of neural network models of this kind.

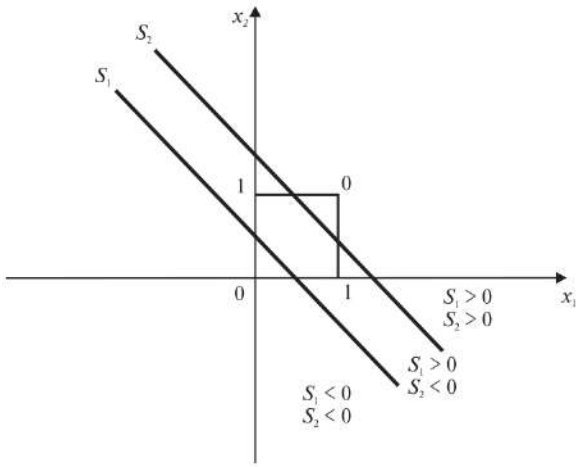

Figure 3. Solving the "EXCLUSIVE OR" problem [11]

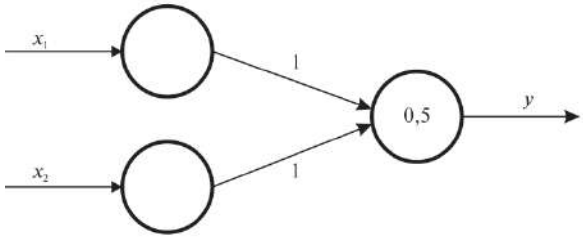

Figure 4. A scheme of a single-layer perseptron solving the "EXCLUSIVE OR" problem [11]

The specification of agents corresponding to the specified actions sets an agent-oriented model for interpreting artificial neural networks. The implementation of this model will be called an artificial neural network interpreter.

Let us consider an example of a description in the neural network method representation language in SCP, that solves the problem, which is formulated as follows: calculate the result of the "EXCLUSIVE OR" logical operation for the values of two logical variables. In Figure 3, the solution to this problem using a signal function is shown.

In the work [11], a single-layer perseptron that solves the problem is described. The perseptron consists of two input neurons and one output neuron, with a given threshold of 0.5 and a signal activation function:

$$F(S) = \begin{cases} 1, 0 < S < 0, \\ 0, \ else \end{cases}$$

The weight coefficients of the input layer synapses are equal to 1. In Figure 4, a scheme of the perseptron is demonstrated.

This perseptron corresponds to the method represented in the ostis-system knowledge base in the neural network

```
proc_exclusive_or_ann
<- scp_method;
<- perceptron;
-> rrel_key_sc_element: _process1;;

proc_exclusive_or_ann = [*
_process1
_<- scp_process;
_-> rrel_1:: rrel_in:: _input_vector;
_-> rrel_2:: rrel_out:: _output_vector;

_<= nrel_decomposition_of_action:: _... (*

        _-> rrel_1:: _...operator1
        (*
                _<- action_calculate_weighted_sum_of_all_neurons_of_layer;;
                _-> rrel_1:: rrel_fixed:: rrel_scp_var:: rrel_input_vector:: _input_vector;;
                _-> rrel_2:: rrel_fixed:: rrel_scp_const:: rrel_synopsis_weight_matrix:: ...
                        (*
                                <- matrix;;
                                -> rrel_1: ...
                                (*
                                        <- number_oriented_set;;
                                        => nrel_oriented_set_string_representation: [1, 1];;
                                *);;
                        *);;
                _-> rrel_3:: rrel_assign:: rrel_scp_var:: rrel_result:: _weighted_sum_vector;

                _=> nrel_goto:: _...operator2;;
        *);;

        _-> _...operator2
        (*
                _<- action_calculate_activation_function_of_all_neurons_of_layer;;
                _-> rrel_1:: rrel_fixed:: rrel_scp_var:: _weighted_sum_vector;;
                _-> rrel_2:: rrel_fixed:: rrel_scp_const:: rrel_threshold_set:: ...
                        (*
                                <- number_oriented_set;;
                                => nrel_oriented_set_string_representation: [0.5];;
                        *);;
                _-> rrel_3:: rrel_fixed:: rrel_scp_const:: rrel_activation_fun:: signal_fun;;
                        (*
                                <- signal_activation_function;;
                                => nrel_definition: signal_function_1_def;;
                        *);;
                _-> rrel_4:: rrel_assign:: rrel_scp_var:: _output_vector;;

                _=> nrel_goto:: _...operator3;;
        *);;

        _-> _...operator3 (* <- return;; *);;
*);;
*];;
```

Figure 5. A method that solves the "EXCLUSIVE OR" problem represented in the neural network method representation language in SCP

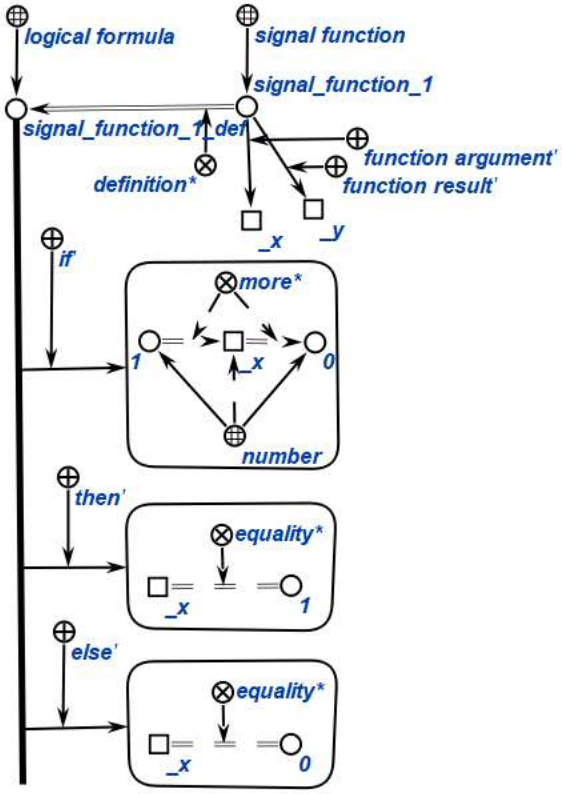

Figure 6. Representation of the activation signal function in the ostis-system memory

method representation language in SCP. This method is represented in Figure 5.

The description of the method consists of a sequence of two generalized action specifications – action for calculating the weighted sum of all neurons of the layer and action for calculating the activation function of all neurons of the layer.

The signal activation function used in the perseptron is defined in the ostis-system memory by the logic formula shown in Figure 6.

Any agent interpreting actions with arguments given with the *activation function'* relation must use an interpreter of mathematical functions that can be used as activation functions. A classification of such functions is shown in [10].

V. Intelligent framework for building neural network methods

The presence of a language for representing neural network methods and its interpreter in SCP allows for the interpretation of the neural network method in the ostis-system memory. The presence in a common memory of not only instances of methods but also concepts that describe them, creates the basis for automating the process of building (designing and training) neural network methods. The ostis-system memory stores knowledge about the methods of which class can solve the problem of a given class, but instances of this method class may not be represented in the system. In this case, the system should be able to inform the user about the possibility of a solution, for which, however, it is necessary to load a certain method into the system. Since the system stores the problem and the requirements for the method of solving it in a common memory, it becomes possible to design the necessary method. This requires the presence of a design framework for the methods of the corresponding classes. In the case of the neural network method, we are talking about an intelligent framework for building neural network methods.

The intelligent framework for creating neural network methods is based on corresponding hierarchies of actions, problems, and methods for building ANN. The presence of such a hierarchy will make it possible to describe the method representation language for building ANN and develop an interpreter for that language.

Creation of the hierarchy of the corresponding actions of building ANN should be studied by the stages of design and training of ANN, which, in the general case, are performed by all the developers of ANN:

1. Problem definition.

The problem definition includes a description of the input data (images / video, time series, text), output data, and requirements for the solution method (speed, memory costs, etc.). It also describes additional information that can help in constructing a problem-solving method (for

example, the specification of the training dataset, if it exist). Usually, at this stage, the developer determines the class of the problem, forms the requirements for the training dataset, if it is not provided.

The execution of this stage by the ANN design framework involves performing the following actions:

- *Action of problem condition translation*. The action translates the description of the problem specified using the ostis-system interface (for example, natural language interface) into the ostis-system memory. The action is required when the problem condition is specified by the user. It is necessary to understand that the problem description goes into the knowledge base not only from the user interface. For example, a problem can be formulated by the system itself in the duration of its life. This action is common to all ostis-systems, so its consideration goes beyond the consideration of the process of building an ANN intelligent design framework.
- *Action of problem classification*. The action determines the class of the problem (problem of regression, detection, clustering, etc.) based on the description of the problem in the knowledge base.
- *Action of finding a suitable training dataset*. The knowledge base can store a set of dataset specifications to which the ostis-system has access. The action searches for datasets that can be used as a training dataset.
- *Action of generating requirements for the training dataset*. If the training dataset was not provided and was not found, then it is necessary to form a description of the requirements for the training dataset, which can be translated into the user interface language and request the necessary dataset from the user.

2. Dataset preprocessing: cleanup

At this stage, features that have incorrect values are detected (for instance, for some examples, the value of the feature may have an undefined value, or a value that does not match in type, or an abnormally large, or very small value). For features that have an undefined value, various elimination methods can be applied, for example, such values can be replaced by the average value of this feature calculated over all examples (for unsequential data), or they can be replaced by average values from adjacent examples (in the case of sequential data), or some fixed value. A radical method for solving the problem is the removal of examples that have undefined feature values from the dataset. However, it is better to use it if there are few examples with missing feature values. Similar strategies are used for outliers and anomalies (but only if the goal is not to predict these anomalies).

In an intelligent design framework, this stage corresponds to the execution of the *action of dataset cleanup*, which is performed in the case of processing a dataset that was not previously represented in the system memory (for example, was received from the user). The implementation of the interpreter (agent) of this action requires the description in memory of the classification of data cleaning strategies and the implementation of methods for applying these strategies.

3. Dataset preprocessing: identifying meaningful features

Engineering of features is implemented, consisting in the selection of features that affect the output of the model; non-meaningful features that do not correlate with the model output are removed. The purpose of this stage is to reduce the dimensionality of the feature space in order to reduce the influence of the overfitting effect on the model.

To reduce the dimensionality of the feature space, the methods of feature selection and feature extraction can be used.

When selecting features, a subset from the original features is formed (backward selection algorithm, recursive feature elimination algorithm, algorithms using random forests).

When extracting features from a set of features, information is extracted to build a new feature subspace (algorithms using an autoencoder).

In an intelligent design framework, this stage corresponds to the execution of the *action of identifying meaningful features*. The implementation of the interpreter (agent) of this action requires the description in memory of the classification of strategies for reducing the dimensionality of the feature space and the implementation of methods for applying these strategies.

4. Dataset preprocessing: transformation

At this stage, the data is prepared for training. Here, special attention should be paid to the presence of categorical features, most often specified by strings. These features can be nominal and ordinal. To encode ordinal features, a sequential numerical code (1, 2, 3, ...) is most often used. For nominal coding, such a solution is incorrect, since these features are fullright and cannot be compared by a numerical code (for example, gender is 0/1). For nominal features, a direct coding method is used, which consists in creating and using fictitious features according to the number of values of the original one. For example, an attribute of a gender (male, female) is converted into two new features – male and female – with the corresponding values for the existing examples.

Feature scaling involves bringing the feature values to one common interval – this is especially relevant for features that have disproportionate means across all dataset – for example, one feature has an average value of 10.000 and another – 12. This can result in minimizing only by feature with the highest values and poor convergence of the training method. Most often, scaling corresponds to performing normalization on an

interval (min-max normalization):

$$x^i_{norm} = \frac{x^i - x_{min}}{x_{max} - x_{min}}$$

where x^i is the value of the feature for a single example i, x_{min} is the smallest value for the feature, x_{max} is the largest value for the feature.

Another scaling technique is to apply feature standardization:

$$x^i_{std} = \frac{x^i - \mu(x)}{\sigma(x)},$$

$\mu(x)$ is a sample mean of a single feature, $\sigma(x)$ is the standard deviation.

Standardization preserves useful information about outliers in the original data and makes the learning algorithm less sensitive to them.

Discretization is used to move from an objective feature to an ordinal one by encoding intervals with a single value (for example, if a feature reflects a person's age, then values can be discretized with the selection of certain age groups, where each group will be encoded by one integer).

In the intelligent design framework, this stage corresponds to the execution of the *action of dataset transformation*. The implementation of the interpreter (agent) of this action requires the description in memory of the classification of methods for scaling features and the implementation of methods for applying these strategies.

5. Dividing the common dataset into training, validation, and test (control) datasets

The entire dataset is divided into training, test, and, in some cases, validation datasets.

The validation set is used to evaluate the impact of changing hyperparameters on the learning outcome and can be used as an additional tool for this along with grid search.

The split is carried out in a ratio of 3:1:1, in percent (60/20/20), if the validation dataset is not used, then 80/20.

In an intelligent design framework, this stage corresponds to the execution of the *action of dataset splitting*.

All previous steps were applied to the dataset; the subsequent steps refer to the used ANN models.

6. Choosing a class of neural network methods in accordance with the formulated problem

At this stage, the selection of the main ANN architecture, which will be used in training, is carried out. However, it should be noted that this selection is relatively conditional; the researcher is not limited to using only one type of ANN to solve a problem (like, for example, a convolutional network for images, since images can also be processed with a conventional multilayer perceptron). Rather, it is about the recommended architecture, but this does not exclude the usage of any other variants of architectures and their combinations within the same model).

Examples of such recommendations are:

- images/video – convolutional neural networks;
- time series – multilayer perceptrons or recurrent networks;
- text information – multilayer perceptrons or recurrent networks;
- sets of characteristics of some objects (for example, car specifications) – multilayer perceptron.

In an intelligent design framework, this stage corresponds to the execution of the *action of selecting a class of neural network methods*.

7. Formation of specifications for input and output data

Additional data transformations are performed related to changing storage structures (for example, converting a multidimensional array to the one-dimensional array, converting types).

In the intelligent design framework, this stage corresponds to the execution of the *action of forming the specification for ANN inputs and outputs*.

8. Selection of optimization method

As part of the work [10], following optimization methods are described:

- stochastic gradient descent (SGD);
- Nesterov method;
- adaptive gradient (AdaGrad);
- adaptive moment estimation (Adam);
- root mean square spread (RMSProp).

In the intelligent design framework, this stage corresponds to the execution of the *action of optimizing method selection*.

9. Selection of an error function to be minimized

At this stage, the error function is set, which will be minimized. For example, MSE is better suited for regression and clustering problems, CE – for classification problems. In the article [10], the classification of such functions within the SD of ANN if described.

These functions are defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widetilde{Y}_i)^2$$

where n is the size of the dataset, Y_i is the reference value of the function, $\widetilde{Y}_i$ is the output obtained by the NN.

$$CE = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \log(\widetilde{Y}_i) + (1 - Y_i)\log(1 - \widetilde{Y}_i))$$

(case of 2-class classification)

$$CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{M} Y_i^c \log \widetilde{Y}_i^c$$

(case of multi-class classification)

In the intelligent design framework, this stage corresponds to the execution of the *action of selecting the error function to be minimized*.

10. Initialization of neural network parameters

In the work [10], within the SD of ANN, methods of primary initialization of ANN have already been described. The most commonly used options for initializing neural network weights and thresholds include:

- initialization with values from a uniform distribution over some small interval, for example, [-0.1, 0.1];
- initialization with values from the standard normal distribution;
- Xavier initialization [12].
  It is used to prevent a sharp decrease or increase in the output values of neurons after applying the activation function during the direct passage of the image through a deep neural network. In fact, initialization by this method is carried out by choosing values from a uniform distribution on the interval $[-\sqrt{6}/\sqrt{n_i + n_{i+1}}, \sqrt{6}/\sqrt{n_i + n_{i+1}}]$, where n_i is the number of incoming connections to this layer and n_i is the number of outgoing connections from this layer. Thus, initialization by this method is carried out for different layers of the neural network from different intervals.
- Initialization obtained from the pre-trained model.
  An initialization option that involves using a pre-trained model as a "starting" model, taken from some repository of pre-trained models, trained by the researcher or during the work of an intelligent system.
- Kaiming initialization [13].
  This initialization method is used to solve the problem of "vanishing" gradient and "exploding" gradient. It is performed by selecting values from a uniform distribution on the interval $[-\sqrt{2}/\sqrt{(1+a^2)fan}, \sqrt{2}/\sqrt{(1+a^2)fan}]$, where a is the angle of inclination to the abscissa for the negative part of the ReLU-type activation function (for a common ReLU function, this parameter is 0), fan is the operating mode parameter, which for the forward propagation phase is equal to the number of incoming connections (to eliminate the effect of the "exploding" gradient), and for the backward propagation phase, to the number of outgoing ones (to eliminate the effect of the "vanishing" gradient).

In an intelligent design framework, this stage corresponds to the execution of the *action of initializing ANN*.

11. Selection of ANN hyperparameters

In practice, some hyperparameters (such as the number of layers, their types, the number of neurons in a layer) are often determined experimentally in the process of iterative search for the best solution to the problem. Although there are ways to partially automate this process, they are still designed for the presence of some preconditions for conducting an experiment, in particular, intervals for changing a parameter (for example, learning rate).

Hyperparameters selected at this stage include:

- ANN training parameters (learning rate, momentum parameter, mini-batch size);
- ANN model architecture that is based on previously formulated specifications of input and output data (for example, the number of neurons in a particular layer(-s) or configurations of entire layers).

Finding the optimal hyperparameters can be obtained, for example, using the grid search method, which allows checking the hyperparameter values taken with a certain step or from a certain interval (tuple). Using this method, the optimal set of hyperparameters is selected, which gives the best results; it is used for subsequent additional training. Otherwise, if the results obtained are acceptable, the further learning process is not carried out at all. The cost of this method should be noted, since, in fact, the searching for different values of training parameters is carried out. To reduce the amount of work, a random search method is used.

To optimize the architecture, the types of layers of the neural network, the number of neurons in each layer, their characteristics are determined – the activation function, for convolutional elements – the size of the kernel, as well as the padding parameter and the convolution step (stride). Here, not only the user version of the network can be evaluated but also the pre-trained architecture. The main rule when selecting is that the number of model parameters should not exceed the size of the training dataset. For pre-trained architectures, this restriction is removed.

In an intelligent design framework, this stage corresponds to the execution of the *action of ANN hyperparameter selection*. The action uses the classification and specification of ANN hyperparameters (described within the SD of ANN [10]).

12. Training the model on the dataset

The model is trained until the selected accuracy is achieved (evaluated on the test dataset) or according to other specified criteria (achievement of the specified number of training epochs, invariability of accuracy over the specified number of epochs, drop in accuracy on the validation dataset, etc.).

Training algorithms have already been described in the SD of ANN [10]. Let us demonstrate their classification:

method of training ANN
⊂ *method*
⊃ *method of training with a teacher*
⇒ *explanation**:
[***method of training with a teacher*** is a method of training using the set target variables **]**

⊃ *method of backward propagation of errors*
:= [MBPE]
⇒ *explanation**:
[MBPE uses a certain optimization method and a certain loss function to implement the phase of backward propagation of the error and change the configurable ANN parameters. One of the most common optimization methods is the method of stochastic gradient descent.]
⇒ *explanation**:
[It should also be noted that despite the fact that the method is classified as one of the methods of training with a teacher, in the case of using MBPE for training autoencoders, in classical publications, it is considered as a method of training without a teacher, since in this case there is no marked data.]

⊃ *method of training without a teacher*
⇒ *explanation**:
[***method of training without a teacher*** is the method of training without using the set target variables (in the self-organization mode)]
⇒ *explanation**:
[When performing the algorithm of the method of training without a teacher, useful structural properties of the set are revealed. Informally, it is understood as a method for extracting information from a distribution, the dataset for which was not manually annotated by a human [14].]

In the intelligent design framework, this stage corresponds to the execution of the *action of training ANN*. An example of formalization of this action is shown in Figure 7

13. Evaluating the ANN effectiveness

After training, the resulting model is evaluated using quality assessment metrics.

Further, the result of the evaluation can be visualized with the confusion matrix and the ROC-curve.

The confusion matrix is a matrix (Fig. 8) that contains information about the number of true positive, true negative, false positive, and false negative classifier predictions.

The ROC-curve is a graph in which, based on the given threshold of the classifier solution, the shares of false positives and true positives are calculated. Based on the ROC-curve, the AUC-indicator (area under the curve) is

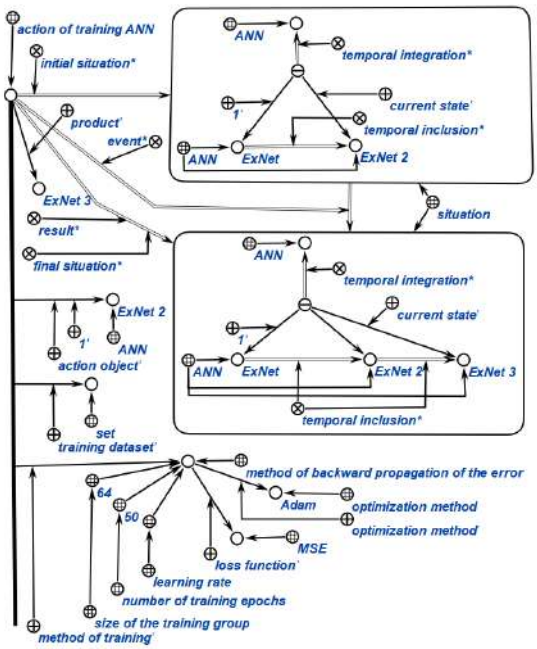

Figure 7. An example of the formalization of the action for training the artificial neural network in the knowledge base [10]

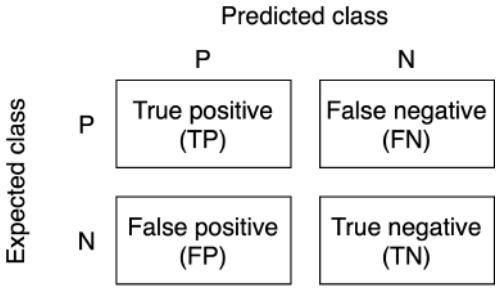

Figure 8. A confusion matrix

calculated, which is used as a characteristic of model quality.

In the intelligent design framework, this stage corresponds to the execution of the *action of ANN performance evaluation*.

Let us consider an example of performance of the described stages by a developer for a specific problem – *classification of digits from the MNIST dataset of handwritten digits*:

1. The initial data of the problem is: a dataset of 70.000 images, pre-divided into a training (60.000 images) and test (10.000 images) datasets. Each image is represented by a two-dimensional array of 28X28 items from the range [0, 255], the numbers represent a shade of gray. In addition, each image has a class label corresponding to a specific digit from 0 to 9.

The problem is: *train a model that will take a two-dimensional array of data as input and return a class*

label corresponding to the recognized digit.

Thus, the type of problem to be solved is **classification**, the nature of the problem data is **images**.

2. There are no anomalies, erroneous data, or features with missing values in this dataset.

3. In the dataset, there are no non-content features.

4. As a method of data preprocessing we use features scaling, min-max normalization on the interval [0, 1].

5. Let us perform the partition of the train dataset into train and validation datasets at a ratio of 4:1 (48.000 examples in the train dataset and 12.000 examples in the validation dataset).

6. Since the dataset includes images, we will use a convolutional neural network.

7. Formation of specifications for input and output data is not required.

8. We will use the stochastic gradient descent (SGD) method as the optimization algorithm.

9. Since the classification problem is being solved, let us select the cross-entropy loss function as the minimizing function.

10. We will use the Kaiming initialization for the network parameters initialization.

11. In stage 6, it was determined that a convolutional neural network would be used to solve the problem. When using one-hot coding, the last full-connected layer will have 10 neurons according to the number of classes in the problem.

For simplicity, we will use the architecture shown in Fig. 9, which does not contain intermediate layers.

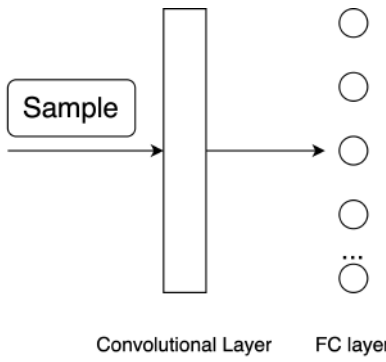

Figure 9. ANN architecture for solving the problem of digit classification

To find the optimal set of hyperparameters, we will apply a random search method.

Let us list the tuples from which hyperparameters will be sampled:

- learning rate – (0.9, 0.1, 0.01, 0.001);
- number of neurons in the convolutional layer – (5, 10, 15, 20);
- size of the convolutional kernel – (3, 5, 7, 9);
- momentum parameter – (0, 0.5, 0.9);
- mini-batch size – (16, 32, 64, 128).

After determining these parameters and evaluating the effectiveness of the algorithm, we obtain the following table:

Table I
PROBLEM RESULTS
(ABBREVIATIONS USED: MBS – MINI-BATCH SIZE, KS – KERNEL SIZE, LR – LEARNING RATE, CNC – CONVOLUTIONAL NEURONS COUNT, ACC – ACCURACY, IT – ITERATIONS COUNT)

#	mbs	ks	lr	momentum	cnc	acc	it
1	128	3	0.001	0.5	10	0.9033	10
2	64	9	0.9	0	15	0.1039	1
3	32	3	0.01	0.5	20	0.9741	10
4	32	7	0.01	0.5	15	0.9794	10
5	16	9	0.001	0.5	20	0.9189	2
6	64	3	0.1	0.5	10	0.9736	10
7	64	7	0.001	0.9	15	0.9007	1
8	32	9	0.1	0.5	5	0.9806	10
9	128	5	0.1	0.5	20	0.98	10
10	32	9	0.01	0.9	5	0.9806	10
11	128	3	0.001	0.9	10	0.893	1
12	32	5	0.9	0.9	20	0.1008	1
13	16	9	0.9	0.5	20	0.0976	1
14	32	7	0.9	0.9	15	0.0932	1
15	128	5	0.01	0.5	20	0.9197	2
16	16	3	0.001	0.5	10	0.904	1
17	16	9	0.001	0	20	0.8866	1
18	128	9	0.1	0.5	5	0.9793	10
19	128	3	0.001	0	10	0.6697	1
20	16	3	0.1	0	15	0.9729	4
21	32	7	0.9	0.5	15	0.1048	1
22	128	7	0.9	0	15	0.1113	1
23	64	9	0.01	0.5	10	0.9482	2
24	16	7	0.9	0	20	0.0985	1
25	16	3	0.1	0.5	5	0.9558	2
26	64	7	0.01	0.9	15	0.9839	10
27	16	7	0.1	0	10	0.9836	10
28	16	5	0.01	0	20	0.9608	2
29	16	5	0.01	0.9	20	0.9847	10
30	32	5	0.01	0.5	15	0.9532	2

It can be noted that the best result (acc = 0.9839) for generalization ability in the validation dataset was obtained with the following parameters: mbs = 64, ks = 7, lr = 0.01, momentum = 0.9, cnc = 15.

12. As a stopping criterion, we selected the simplest one on reaching a given number of epochs of training. No pre-training was performed, and the model obtained after the hyperparameter fitting procedure was used to estimate the generalization ability. The generalization ability on the test dataset was **0.9853**, i.e. **98.53%**.

13. By constructing a confusion matrix based on the trained model and the test dataset, we obtain the result illustrated in Fig. 10

The obtained matrix is diagonally dominant, so the resulting model does relatively few errors.

Based on the analysis of the stages of constructing the ANN that developers perform, the following classification of actions for the construction of ANN can be derived:

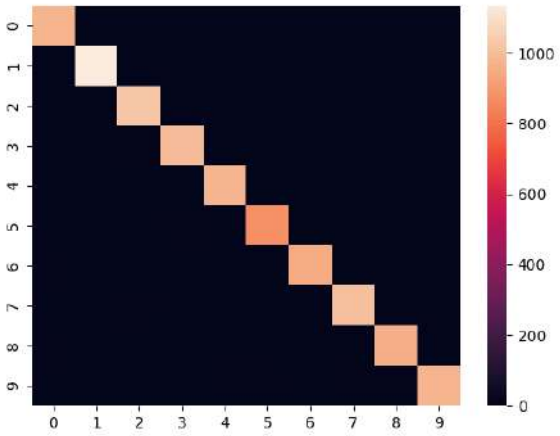

Figure 10. A confusion matrix for the MNIST problem

action of building ANN
⇒ *decomposition**:
{• *action of dataset processing*
⇒ *decomposition**:
{• *action of finding a suitable training dataset*
• *action of generating requirements for the training dataset*
• *action of dataset cleanup*
• *action of identifying meaningful features*
• *action of dataset transformation*
• *action of dataset splitting*
}
• *action of designing ANN*
⇒ *decomposition**:
{• *action of selecting a class of neural network methods*
• *action of forming the specification for ANN inputs and outputs*
}
• *action of training ANN*
⇒ *decomposition**:
{• *action of optimizing method selection*
• *action of selecting the error function to be minimized*
• *action of initializing ANN*
• *action of ANN hyperparameter selection*
• *action of ANN performance evaluation*
}
}

The implementation of the interpreter of actions for building ANN considered in this section and the description in the knowledge base of the expert knowledge of the ANN developers (and thus the implementation of the intelligent ANN design framework) will automatically, based on the problem description, generate neural network methods in the ostis-system memory, which is one of the key directions for development of this work.

VI. Conclusion

In this article, an approach to the integration and convergence of artificial neural networks with knowledge bases in next-generation intelligent computer systems through the representation and interpretation of the artificial neural network in the knowledge base is described.

The syntax, denotational, and operational semantics of the neural network methods representation language are described, which allows representing and interpreting any ANN in the memory of the intelligent system. The existence of such language generates semantic compatibility of neural network method with other methods represented in the system memory, which allows analyzing the ANN itself and its performance stages by any other methods of the system.

The availability of neural network representation language allows describing the expert knowledge of the developers of the information network in the system memory. In this article, the stages of building ANN, which are carried out by the developers of ANN, are represented. Based on these stages, in order to design an intelligent framework for building neural network methods, the actions of building the neural network methods has been classified and described in the knowledge base.

The design and implementation of the intelligent framework for building ANN in the knowledge base of the system is one of two main directions for further development of this work.

The second main direction is to develop an approach to the processing of fragments of the knowledge base by ANN. For this direction, it is necessary to develop a universal algorithm of mutual-ambiguous matching of knowledge base fragments and input vectors of ANN. A knowledge representation language is able to represent any knowledge. The presence of a neural network method in the system, which is able to take knowledge fragments on the input, will allow solving new, poorly studied classes of problems.

Acknowledgment

The authors would like to thank the research groups of the Departments of Intelligent Information Technologies of the Belarusian State University of Informatics and Radioelectronics and the Brest State Technical University for their help in the work and valuable comments, in particular, Vladimir Golenkov and Daniil Shunkevich.

References

[1] V. V. Golenkov, N. A. Gulyakina, D. V. Shunkevich, *Open technology for ontological design, production and operation of semantically compatible hybrid intelligent computer systems*, G. V.V., Ed. Minsk: Bestprint, 2021.

[2] D. Castelvecchi, "Can we open the black box of AI?" *Nature News*, vol. 538, no. 7623, Oct 2016.

[3] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," 2016. [Online]. Available: https://arxiv.org/abs/1602.04938

[4] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

[5] T. R. Besold, A. d'Avila Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kuehnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, and G. Zaverucha, "Neural-symbolic learning and reasoning: A survey and interpretation," Nov. 2017, (accessed 2020, Jun). [Online]. Available: https://arxiv.org/pdf/1711.03902.pdf

[6] A. d'Avila Garcez, T. R. Besold, L. de Raedt, P. Földiak, P. Hitzler, T. Icard, K.-U. Kühnberger, L. C. Lamb, R. Miikkulainen, and D. L. Silver, "Neuralsymbolic learning and reasoning: Contributions and challenges," *In: McCallum, A., Gabrilovich, E., Guha, R., Murphy, K. (eds.) Proceedings of the AAAI 2015 Propositional Rule Extraction under Background Knowledge 11 Spring Symposium on Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches. AAAI Press Technical Report*, vol. SS-15-03, 2015.

[7] A. Kroshchanka, V. Golovko, E. Mikhno, M. Kovalev, V. Zahariev, and A. Zagorskij, "A Neural-Symbolic Approach to Computer Vision," in *Open Semantic Technologies for Intelligent Systems*, V. Golenkov, V. Krasnoproshin, V. Golovko, and D. Shunkevich, Eds. Cham: Springer International Publishing, 2022, pp. 282–309.

[8] V. Golenkov, N. Gulyakina, I. Davydenko, and D. Shunkevich, "Semanticheskie tekhnologii proektirovaniya intellektual'nyh sistem i semanticheskie associativnye komp'yutery [semantic technologies of intelligent systems design and semantic associative computers]," *Open semantic technologies for intelligent systems*, pp. 42–50, 2019.

[9] D. Shunkevich, "Ontology-based design of hybrid problem solvers," in *Open Semantic Technologies for Intelligent Systems*, V. Golenkov, V. Krasnoproshin, V. Golovko, and D. Shunkevich, Eds. Cham: Springer International Publishing, 2022, pp. 101–131.

[10] M. Kovalev, "Ontology-Based Representation of an Artificial Neural Networks," in *Open Semantic Technologies for Intelligent Systems*, V. Golenkov, V. Krasnoproshin, V. Golovko, and D. Shunkevich, Eds. Cham: Springer International Publishing, 2022, pp. 132–151.

[11] V. A. Golovko and V. V. Krasnoproshin, *Nejrosetevye tekhnologii obrabotki dannyh*. Minsk : Publishing House of the BSU, 2017.

[12] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015. [Online]. Available: https://arxiv.org/abs/1502.01852

[14] J. Goodfellow, I. Benjio, and A. Courville, *Deep learning*. Moscow : DMK Press, 2017.

Конвергенция и интеграция искусственных нейронных сетей с базами знаний в интеллектуальных компьютерных системах нового поколения

Ковалёв М.В., Крощенко А.А., Головко В.А.

В статье рассмотрен подход к интеграции и конвергенции искусственных нейронных сетей с базами знаний в интеллектуальных компьютерных системах нового поколения с помощью представления и интерпретации искусственных нейронных сетей в базе знаний. Описаны синтаксис, денотационная и операционная семантика языка представления нейросетевых методов в базах знаний. Описаны этапы построения нейросетевых методов решения задач с помощью интеллектуальной среды проектирования искусственных нейронных сетей.

Received 14.11.2022