Algorithms for neurocomputer processing and storage of artificial neural networks

V A Romanchuk

1Ryazan State University named for S.Yesenin, 46, Svobody Ave., Ryazan, 390000, Russia
E-mail: v.a.romanchuk@365.rsu.edu.ru

Abstract. We suggest some general principles and a universal open format for processing and storing artificial neural networks to be subsequently processed on the basis of neurocomputer devices of various architectures. To enhance this universalization, we choose to employ the ONNX format (Open Neural Network Exchange) and the Wolfram Language framework. In 2018, such repository of artificial neural networks was implemented. For functional convenience, storage of artificial neural networks and for using the web version, we chose the JSON technology.

1. Introduction
Currently, due to growing high-performance computing and new architectures of artificial neural networks, we observe rapid development of artificial intelligence systems for solving problems of processing signals, images, video, as well as control, forecasting, optimization, etc. to optimize work with these, special frameworks and libraries have been developed such as: Caffe, Torch, MXNet, TensorFlow, etc.

However, there is the issue of efficient storage for neural networks, since when working with neural networks and appropriate processing algorithms, the complexity most often lies in the size and heterogeneity of the network data. Now each library has a specific format and unique methods for storing artificial neural networks (ANN), which often complicates research and technical advancement in this area.

It should also be noted that the libraries and developed methods do not support processing ANNs that are based on neurocomputer devices, while many companies have recently started manufacture neurochips and neural accelerators. The TensorFlow library is of great interest, as it employs a method of processing ANNs based on GPU devices, but does not provide a uniform format or principles for storing ANNs; it cannot support a high-performance neuro-computer hardware base. In 2018, Wolfram Research undertook developing a single format in order to create its own repository of neural net-works, but (1) the format developed is proprietary, and (2) not intended for use on neurocomputer devices and does not take into account the possibility of using data parallelism or command parallelism. Hence the tasks delineated here remain important: studying ANNs, creating principles and a common format for storing them for subsequent processing of ANNs based on neuro-computer devices, depending on their architecture, similar to the methods used to launch ANNs on the TensorFlow library GPU.

2. The task of neurocomputer processing and storage of artificial neural networks
Let $Z_{INS}^{(j)}$ be a $j$th class of neural network tasks, which is implemented on a neuroprocessor platform as an ANN. The ANN can be specified as a tuple of parameters and properties [7]:

- a set of ANN inputs $Net_X = \{net_{i1},...,net_{i\alpha},...,net_{in}\}$, where each $net_{i\alpha}$ input is characterized by its type and range of allowed values;
- a set of ANN outputs $Net_Y = \{net_{\gamma1},...,net_{\gamma\beta},...,net_{yn}\}$, where each $net_{\gamma\beta}$ output is characterized by its type and range of allowed values;
- a set of neurons $Net_N = \{n_1,...,n_i,...,n_c\}$, where each neuron must be emulated to solve a certain $Z_{INS}^{(j)}$ class of tasks;
- a set of weighting factors $Net_W = \{net_{w1},...,net_{w\mu},...,net_{wnw}\}$, characterized by the types and range of possible values;
- neuron activation function $F$;
- method for setting weighting factors;
- and other parameters.

To implement the task of storing a trained artificial neural network, it is necessary to solve two tasks: storage of ANN architecture and storage of ANN weights obtained during training.

It is important to keep the architecture and weights separate so that weighting factors can be loaded into a network with a different architecture. For example, this approach is used when training without a teacher is combined with teacher-based training. At the first stage, they run training without a teacher using an auto-encoder, a deep belief network, or another method, and then the weights obtained are loaded into a network of another architecture, which is further trained using the standard training approach with a teacher, using the back propagation method. The combination of these two methods allows training the network when there is little marked data for training [7].

3. Format of processing and storage of ANN architecture
The ANN architecture can be saved using existing ANN tools, for example, the JSON format for Keras library. In this case, saving will look like this [8]:

```python
model_json = model.to_json()
jsoan_file = open("mnist_model.json", "w")
jsoan_file.write(model_json)
jsoan_file.close()
```

Yet, for most cases, this format cannot be applied due to differences in data sets and library formats such as Caffe, Torch, MXNet, and TensorFlow. To implement a universal format, we need a representation of an artificial neural network in the form of multidimensional data arrays with selection of parallel data streams and commands.

An ANN topology can be described as a set of connections between neurons that defines the dependencies between ANN neurons, which can be represented as a matrix of neuron connections, inputs and outputs of these ANNs: $Net_E \rightarrow MNet = [MNet_{ij}]$ of a $(nc + xn + yn) \times (nc + xn + yn)$ dimension, $Net_E = \{N_{11},N_{12},...,N_{nc1},...,N_{ncn}\}$. This matrix shows the ANN topology and the number of ANN layers, their present or missing bypass connections, and the transfer functions of neurons.

We introduce the notion of a strict order relationship $\prec$, which is a binary relation that meets the following requirements: anti-reflexivity, anti-symmetry, and transitivity. In this case, $\forall n_i,n_j \in N, n_i \prec n_j$ means that the $n_j$ neuron uses the output of the $n_i$ neuron as input data, so an emulation of the $n_j$ neuron cannot be performed without previous emulation of the $n_i$ neuron.
performed. $\prec$ relationships between neurons are determined by the connection of the ANN with the $Net_{E} = \{N_{11}, N_{12}, \ldots, N_{nc}, \ldots, N_{ncnc} \}$ set of connections.

Let us consider the terms of a strict order relationship, $\prec$:

1. Any $n_{i}$ neuron that reflects a neuron model cannot use its own output data. Thus, the requirement of anti-reflexivity of operations is met.

2. If the output data of an $n_{i}$ neuron are used as input data for an $n_{j}$ neuron, then, based on the ANN topology, the output data of the $n_{j}$ neuron cannot be used as input data for the $n_{i}$ neuron, therefore the condition of anti-symmetry is also met.

3. It is possible that the output data of the $n_{i}$ neuron are used as input data of the $n_{j}$ neuron, and the output data of the $n_{j}$ neuron are used as input data of the $n_{k}$ neuron, thus the condition of transitivity of operations is true.

Each element of the matrix can take these values: $MNet_{ij} = \{', 'n', 'y\}$:

- $'n'$ ('no') – between $n_{i}$ and $n_{j}$ subroutines and $(j > i)$, there is no strict order relationship $(n_{i} \nless n_{j})$;

- $'y'$ ('yes') – between $n_{i}$ and $n_{j}$ subroutines, there is a strict order relationship $n_{i} \prec n_{j}$; $j > i$;

- '$-$' – there can be no strict-order relationship between neurons.

For example, let us consider the matrix of connections for six $N = \{n_{1}, n_{2}, n_{3}, n_{4}\}$ neurons, two $X = \{x_{1}, x_{2}\}$ inputs, and one $Y = \{y_{1}\}$ output with the perceptron topology shown in Figure 1.

In the matrix, all the elements below the main diagonal are equal, since there are no feed-backs in the perceptron.

$$MNet = \begin{bmatrix}
-x_{1} & x_{2} & n_{1} & n_{2} & n_{3} & n_{4} & y_{1} \\
-x_{1} & n & y & y & n & n & n \\
x_{2} & n & - & y & y & n & n \\
n_{1} & n & n & - & n & n & y \\
n_{2} & n & n & n & - & n & y \\
n_{3} & n & n & n & n & - & y \\
n_{4} & n & n & n & n & n & -
\end{bmatrix}$$

Based on the theory of neural networks, the task consists of two subtasks: the subtask of $Z^{(j)}$ ANN learning and the subtask of $Z^{(j)}$ ANN emulation. Most often, the neurochip transmits a neural network...
trained on an external computer to the computer neurochip, in its artificial neural network constructed on the basis of the neurochip. But on the neurodevice also has opportunities for further training by means of the ANN neurochip. Let us introduce a set of operations named \( O = \{O_1, O_2, \ldots, O_i, \ldots, O_{nc}\} \). To solve the \( Z^{(j)} \) problem, all neurons should be emulated in accordance with strict order relations between neurons, following which the \( Z^{(j)} \) class of the task can be represented as a tuple of the neuron type emulation operations. Then the \( Z^{(j)} \) subtask is a data tuple: \( Z^{(j)} = \{\text{Net}_X, \text{Net}_Y, \text{Net}_N, \text{Net}_W, M\text{Net}, \ldots\} \). Taking into account the introduced \( Z^{(j)} \) set of operations, the class of neural network tasks is a tuple consisting of \( O_1, O_2, \ldots, O_m, \ldots, O_{nc} \) operations of the \( j \)-class of the task can be represented as a tuple of \( Z^{(j)} \) operations on the neurochip. Let us introduce the number of emulated ANN neurons, since a certain operation is a mathematical model of a formal neuron: \( O_m = f(\sum_{m=1}^{n} w_m x_m + w_0) \).

For some \( O_i \) and \( O_j \) operations, this \( \prec \) relation is also applicable, defined as follows: \( \forall O_i, O_j \in O \), \( O_i \prec O_j \) as input data, the \( O_j \) operation uses the output data of the \( O_i \) operation, thus the \( O_j \) operation cannot be performed without preliminary performance of the \( O_i \) operation. The \( O_m \) operation can be considered as a set of independent operations based on the connections and functional neurocomputer blocks implemented as: \( O_m = \{O_{m,1}, O_{m,2}, \ldots, O_{m,NP}\} \), where \( NP \) is the number of possible functional stages of neuron emulation operations in the neuroprocessor device. In general, the number of operations for neuron emulation is: \( NP = nx + (nx - 1) + 1 = 2^*nx \), where \( nx \) is the number of neuron inputs based on the ANN topology. The first item defines the operation of multiplying the input value by the weight of the connection. The second term defines the operation of summing the results of the product. The third term is the neuron activation operation. The relations of strict order for individual functional modules of a single neuron emulation operation can be represented as a matrix of dimension:

\[
M\text{Neuron} = \begin{bmatrix}
- x_1 & \ldots & x_a & u_1 & \ldots & u_n & s_1 & \ldots & s_{n-1} & a & y \\
x_1 & - n & n & y & n & n & n & n & n & n & n \\
\ldots & n & - n & n & \ldots & n & n & n & n & n & n \\
x_n & n & n & - n & \ldots & n & n & n & n & n & n \\
u_1 & n & n & n & - n & \ldots & y & n & n & n & n \\
\ldots & n & n & n & - n & \ldots & y & n & n & n & n \\
u_n & n & n & n & n & \ldots & y & n & n & n & n \\
s_1 & n & n & n & n & n & - n & n & n & n & n \\
\ldots & n & n & n & n & n & - n & n & n & n & n \\
s_{n-1} & n & n & n & n & n & n & - y & n & n & n \\
a & n & n & n & n & n & n & n & n & n & - y \\
y & n & n & n & n & n & n & n & n & n & n
\end{bmatrix}
\]

Then the ANN shown in Figure 1 can be represented as a set of parallel operations (Figure 2).
Then, for some $Z^{(j)}$ class of problems, the $MNetA$ matrix describing all the strict order relations between operations can be obtained, as a combination of $MNeuron$ and $MNet$ matrices, where each matrix element can be obtained as a $\gamma$ map, i.e., can be displayed as an $MNeuron$ matrix: $\gamma : MNet_{ij} \rightarrow MNeuron$. The strict order $MNet$ matrix relations are preserved relative to the input and output values of the $MNeuron$ matrix. The $MNetA$ matrix has the following dimensions: $(nc + xn + yn) \times (2*xn + (xn - 1) + 2) \times (nc + xn + yn) \times (2*xn + (xn - 1) + 2)$. For the example above, the $MNetA$ matrix will have the following dimension and can be represented as follows:

$$MNetA = \begin{bmatrix}
-x_1 & x_1 & \ldots & x_n
-u_1 & u_1 & \ldots & u_{n+1}
-s_1 & s_1 & \ldots & s_{n-1}

-a_1 & a_1 & \ldots & a_{n-1}

-y_1 & y_1 & x_2 & \ldots

\vdots & \vdots & \ddots & \vdots
\end{bmatrix}$$

4. Format of processing and storage of ANN architecture

To store weighting formats, we can use many file formats: csv, JSON, HDF5, a relational database, or others. The most common format nowadays is HDF5, which is similar to a hierarchical file system; and its paths are similar to the POSIX syntax used to access data; corresponding metadata are stored as sets of named object attributes.

It matters how much memory is needed to store all the weights in the trained neural network. Let us consider the most common ANN topologies and describe the amount of stored and transmitted data for these (data transmitted during parallelization of the data stream).
1. Topology of a single-layer perceptron. The required amount of memory can be calculated as follows: \( K_O(SLP) = \log_2(\max(\text{net}_{\text{out}})) \cdot nw = \log_2(\max(\text{net}_{\text{out}})) \cdot xn \cdot nc \). The total amount of data to be transmitted is equal to: \( K_B(SLP) = \sum_{i=1}^{nc} \log_2(\max(\text{net}_{\text{out}})) + \sum_{l=1}^{2^{q-1}} \sum_{j=1}^{nc_i} \log_2(\max(f_j(\sum_{i=1}^{n} w_{ij} x_i + w_0))) \).

2. The topology of a multilayer perceptron. The required amount of memory can be calculated as follows: \( K_O(MLP) = \log_2(\max(\text{net}_{\text{out}})) \cdot (xn \cdot nc_i + \sum_{l=1}^{Sl-1} nc_i \cdot nc_i^{l+1}) \). The amount of data transferred is equal to: \( K_B(MLP) = \sum_{i=1}^{nc} \log_2(\max(\text{net}_{\text{out}})) + \sum_{l=1}^{2^{Sl-1}} \sum_{j=1}^{nc_i} \log_2(\max(f_j(\sum_{i=1}^{n} w_{ij} x_i + w_0))) \).

5. Conclusion
As a result of this work, a universal format and general storage principles for ANNs of various topologies were developed in order to create a public repository of artificial neural networks presented in a single format. We have developed a method, general principles and a common format for processing and storing ANNs, taking into account the possibilities of data parallelism and command parallelism, and their subsequent processing adapted to specific hardware based on neurocomputer devices, which allows building a base for creating an open and accessible repository of artificial neural networks, including trained and untrained ANN models from different sources, for solving such various problems as: classification, image processing, speech recognition and others.

6. Acknowledgments
The reported study was funded by RFBR according to the research project №18-47-623001 r_mol_a and the government of the Ryazan region”.

References
[1] Barabash O, Kravchenko Y, Mukhin V, Kornaga Y and Leshchenko O 2017 International Journal of Intelligent Systems and Applications(IJISA)9 pp 1–9
[2] Kryuchkovsky V V and Usov A V 2013 Proceedings of Odessa National Polytechnic University2154-160
[3] Boyd S, Parikh N, Chu E, Peleato B and Eckstein J 2011 Foundations and Trends in Machine learning3: 1-122.
[4] Bottou L 2010 Proceedings of COMPSTAT'2010 (Paris)177–186
[5] Bottou L 2012 Neural networks: Tricks of the trade421–436
[6] Yakkali R T and Raghava N S 2017 International Journal of Image, Graphics and Signal Processing(IJIGSP)9 38–49
[7] Romanchuk V A 2018 MATEC Web of Conferences177(1)201816103004
[8] Romanchuk V A 2017 IOP Conference Series: Materials Science and Engineering161012033