IEEE Second International Conference on Data Stream Mining & Processing
August 21-25, 2018, Lviv, Ukraine

Implementation of Neural Networks with Help of a Data Flow Virtual Machine

Kostyantyn Kharchenko
System Design Department
Institute for Applied System Analysis
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
k.kharchenko@kpi.ua

Oleksandr Beznosyk
System Design Department
Institute for Applied System Analysis
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
o.beznosyk@kpi.ua

Valery Romanov
System Design Department
Institute for Applied System Analysis
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"
Kyiv, Ukraine
v.romanov@kpi.ua

***Abstract*—The main goal of this paper is to show how a neural network can be implemented with help of the data flow management system at a virtual machine. As an example, the three-layer neural network realization has been investigated to solve a simple XOR function with two inputs and one output. For that purpose, a sigmoid command required to make a neuron activation function has been added into the data flow virtual machine. It is presented in the paper that neural networks can be described as data flows with help of the declarative approach on a base of the JSON format.**

Keywords—neural networks, data flow virtual machine, JSON, activation functions

I. Introduction

Using virtual machines in computational systems is well-known for years. Such an approach established itself well as it supports compatibility at the bytecode level and its productivity on various hardware platforms is often comparable to the software development with native code compilers.

In the same time, the data flow computations were developing actively in contrast to the command flow ones. Currently, data flow management systems are widespread in the different areas.

In the previous papers [1, 2, 3, 4], some implementations of the data flow virtual machine (DFVM) as well as a concept of input data representation for a data flow system were described. One of them is a JSON-based format for the data flow virtual machine that is rather simple and effective. So, at the moment a DVFM's input file is a JSON file that is easily understandable and self-descriptive.

Currently, it is possible to describe neural networks as data flows [5, 6, 7]. The aim of the paper presented is to create an example of the test neural network working at the data flow virtual machine. At the same time, the neural network input configuration is described by a static JSON file in contrast to the widespread approach to describe neural networks by means of the codes in Python, C++, Java etc. programming languages.

The possibility and convenience to describe a neural network in the declarative form at the level of neural network's separate signals is under investigation. The problem of the description at the layer's level will be considered separately.

II. Existing Solutions

There are a lot of tools for neural network computations such as, for example, TensorFlow [8], Theano [9], MXNet [10], CNTK [11], Keras [12] (known as symbolic frameworks), Torch [13, 14], Caffe [15] (imperative frameworks).

TensorFlow is a platform-independent open source software library for artificial intelligence and machine learning developed by Google for internal use to build and train neural networks for automatically finding and classifying images and correlations, with a goal to achieve the quality of human perception. It is currently used for researches as well as the development of Google products such as Speech Recognition, Gmail, Photos, Search, Maps. The main API for working with the library is implemented for Python while there are implementations for C++, Haskell, Java and Go too. TensorFlow computations are represented as stateful data flow graphs. Its name comes from so called "tensors" – multidimensional data arrays, on which such neural networks perform the operations. TensorFlow can run on multiple central and graphic processors (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). In 2015, it was released under the open Apache 2.0 license.

Theano is a Python numerical computations library. It deals with symbolically specified mathematical computations and optimizes them to produce efficient low-level realization. The calculations in Theano are expressed by the NumPy syntax and compiled for efficient parallel processing on conventional CPUs and GPUs. On September 28, 2017, it was announced that work on the project was discontinued after the release of 1.0, while the minimum support was maintained for one year.

Apache MXNet is a modern open-source fast and scalable deep learning framework with a compact and easy-to-use machine learning API. It is used to train and deploy deep neural networks, and it supports a flexible programming model and multiple languages such as C++, Python, Perl, Scala, Matlab, Wolfram, JavaScript, Go, R. MXNet includes the Gluon interface, which makes it easy for developers to get started with deep learning, for example, in the cloud or in mobile applications. With just a few lines of Gluon code, such features as linear regression, convolutional networks, and recurring LSTMs for object and speech recognition, recommendation, and personalization can be developed. The MXNet library is portable. Also, it is scalable to multiple graphic processors and computers. MXNet is supported by major public cloud providers such as Amazon and Microsoft as well as a number of world-famous companies and research institutions.

Microsoft Cognitive Toolkit, formerly known as CNTK, is a deep learning framework developed by Microsoft Research. It describes neural networks as a series of computational steps via a directed graph.

Keras is an open neural network library written in Python. It is an add-on for the Deeplearning4j, TensorFlow and Theano frameworks. It is capable of working on top of them. It is aimed at operational work with deep learning networks, while being designed to be compact, modular and expandable. It was planned that Google will support Keras in the main TensorFlow library, but Keras was designed as an interface rather than an end-to-end machine learning system. It represents a high-level, intuitive set of abstractions, which makes the formation of neural networks easy, regardless of the library of scientific computing used at the lower level. This library contains numerous implementations of widely used building blocks of neural networks, such as layers, target and transfer functions, optimizers, and many tools for simplifying the work with images and text.

Torch is an open library for the open source Lua programming language. It provides a lot of algorithms for deep machine learning and scientific computing. The kernel is written in C, the application part is executed on LuaJIT, also it supports parallelization of calculations by means of CUDA and OpenMP.

Caffe is a system for deep learning developed by Yangqing Jia as part of his doctoral work at the University of California at Berkeley. Caffe is open source software distributed under the BSD license. It is written in C++ and supports the Python interface. Its name comes from the reduction of “Convolution Architecture For Feature Extraction”. Caffe first ported the MATLAB implementation of fast convolutional neural networks (CNN) to C and C++. Caffe includes numerous algorithms and deep-learning architectures for classifying and clustering image data. CNN, R-CNN (recurrent neural network), LSTM (long short-term memory) and fully connected neural networks are supported. With Caffe, the graphics processor-based acceleration can be used with Nvidia's cuDNN. Caffe supports Python and MATLAB programming environments. Yahoo has integrated Caffe into Apache Spark to distribute deep learning.

Some of the systems mentioned operate with data flows but all of them use an imperative form of the neural network description and, thus, require using programming languages to describe a neural network behavior. In the same time, the approach proposed in this article allows describing a neural network by means of the declarative JSON-like constructions; it seems to be more convenient and simple than using conventional programming languages. Each of the approaches mentioned has its own advantages and disadvantages but the main idea is that the declarative approach would allow working with neural networks for users without knowledge of the programming languages.

III. Data Flow Virtual Machine for Neural Network Computations

Let’s consider an example often used for an acquaintance almost with any system or library for neural networks.

Let’s assume that neural network’s coefficients to solve a test XOR example are already known (the problem of neural network training to select coefficients required, for instance, by a gradient descent method is not currently under investigation).

At the current implementation of a neural network, one neuron can accept only two input signals. The XOR function to be implemented on the neural network is presented in Table 1.

TABLE I. XOR Function

A	0	0	1	1
B	0	1	0	1
Z	0	1	1	0

A. *Implementation of Sigmoid Function in DFVM*

Let’s implement a sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

in DFVM. The following DFVM JSON file shows an example of using sigmoid; if the input value is 1.0 then the expected output is 0.731059:

```
{
"comment": "sigmoid y = 1 / (1 + exp(-x))",
"inputs": [ {"name":"x", "value":1.0},
            {"name":"y", "value":0.0} ],
"outputs": [ {"name":"y", "assert":
           [{"equal":0.731059}]} ],
"nodes": [ {"double":"x"}, {"double":"y"} ],
"commands": [ {"code":"sigmoid", "inputs":["x"],
              "outputs":["y"]} ]
}
```

This sigmoid function can be used as an activation one.

For a lot of tasks in neural networks, it is needed also to use w_0 value (bias). Let’s add it to the sigmoid function:

$$y = \frac{1}{1+e^{-x+w_0}}$$

This will allow to simplify working with a neural network and to avoid adding one more DFVM component to implement addition (Fig. 1).

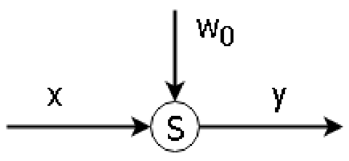

Fig. 1. DFVM implementation of sigmoid function with bias.

So, the DFVM sigmoid function takes two input arguments x and w_0 and looks as follows:

```
{
"comment": "sigmoid y = 1 / (1 + exp(-x+w0))",
"inputs": [ {"name":"x", "value":1.0},
            {"name":"w0", "value":1.0}
            {"name":"y", "value":0.0} ],
"outputs": [ {"name":"y"} ],
"nodes": [ {"double":"x"},
               {"double":"w0"}
               {"double":"y"} ],
"commands": [ {"code":"sigmoid_bias",
               "inputs":["x", "w0"],
               "outputs":["y"]} ]
}
```

B. *Description of a Simple Neural Network with the Data Flow Paradigm*

The following data flow JSON file implements a simple neural network in the data flow virtual machine. The first input layer consists of two neurons, the second (hidden) layer consists of two neurons, and the output layer consists of one neuron (Fig. 2).

This DFVM JSON file represents an implementation of neural network on data flow paradigm for XOR sample shown in Fig. 3.

Each neuron has a w_{ij} coefficient for multiplication of the neuron's input and passes a result to the sigmoid activation

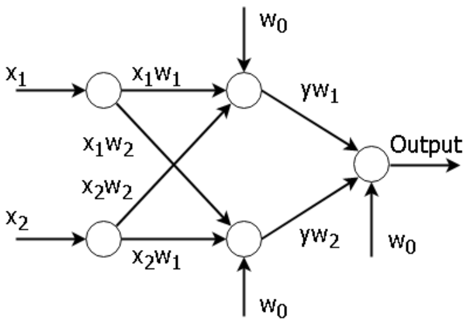

Fig. 2. Simple example of neural network for XOR function.

function. This neural network consists of two inputs *input1*, *input2* and 6 weight coefficients w_{ij} . All these inputs are of *double* type. This neural network consists of two layers, and sigmoids are used as activation functions:

```
{
"comment": "neural network",
"inputs": [ {"name":"input1", "value":1.0},
               {"name":"input2", "value":1.0},
               {"name":"w11", "value":1.0},
               ...
               {"name":"w32", "value":1.0}],
"outputs": [ {"name":"output"} ],
"nodes": [  {"double":"input1"},
               {"double":"input2"},
               {"double":"w11"},
               ...
               {"double":"output"}  ],
"commands": [ {"code":"mul", "inputs":["input1",
               "w11"], "outputs":["o11"] },
               {"code":"mul", "inputs":["input2",
               "w21"], "outputs":["o21"] },
               {"code":"mul", "inputs":["input1",
               "w12"], "outputs":["o12"] },
               {"code":"mul", "inputs":["input2",
               "w22"], "outputs":["o22"] },
               {"code":"add", "inputs":["o11",
               "o21"], "outputs":["s1"] },
               {"code":"add", "inputs":["o12",
               "o22"], "outputs":["s2"] },
               {"code":"sigmoid", "inputs":["s1"],
               "outputs":["o31"] },
               {"code":"sigmoid", "inputs":["s2"],
               "outputs":["o32"] },
               {"code":"mul", "inputs":["o31",
               "w31"], "outputs":["o41"] },
               {"code":"mul", "inputs":["o32",
               "w32"], "outputs":["o42"] },
               {"code":"add", "inputs":["o41",
               "o42"], "outputs":["s3"]},
               {"code":"sigmoid", "inputs":["s3"],
               "outputs":["output"]} ]
}
```

IV. Implementation Details and Other Types of Activation Functions in DFVM

The existing data flow virtual machine did not require a lot of modifications, except for adding new commands for activation functions. A component diagram of DFVM is presented in Fig. 4.

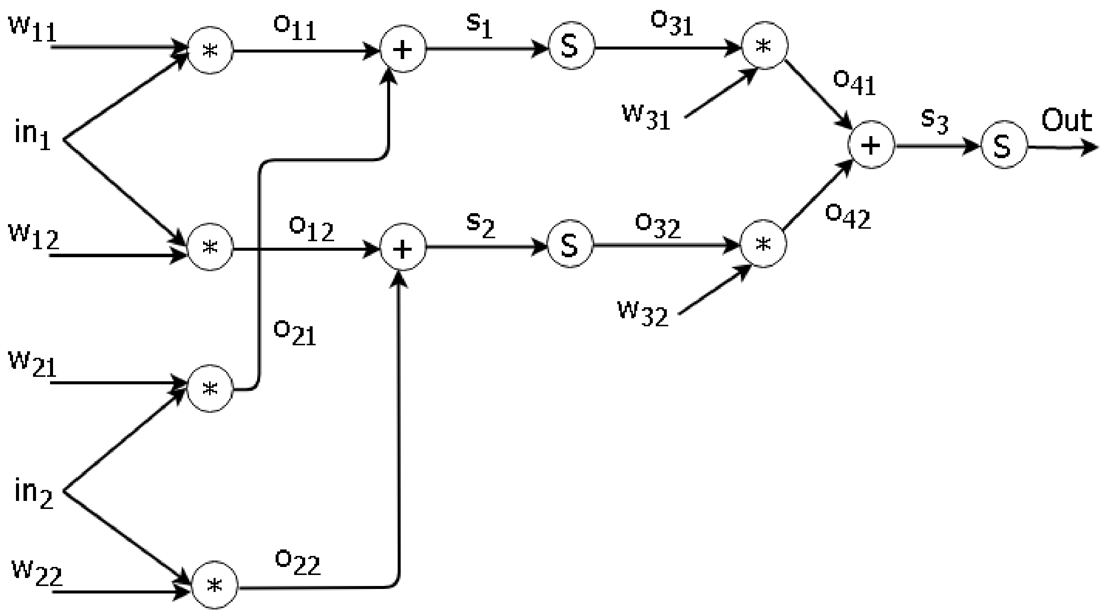

Fig. 3. Neural network on data flow paradigm.

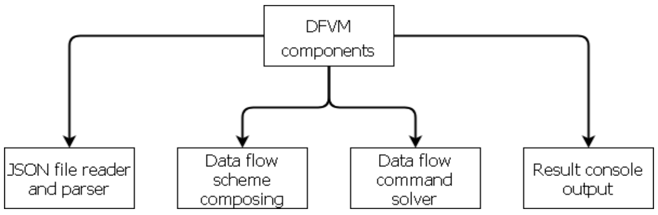

Fig. 4. DFVM component diagram.

At the moment, there are the following activation functions in DFVM:

- relu
- softsign
- sigmoid_bias
- softplus
- sigmoid
- tanh

In C++ code, an activation function implementation takes place in the same way as for usual DFVM commands such as *add*, *sub*, *div*, *mul* for a *double* type. The activation function can take one or more input parameters and return one output parameter. Entire C++ programming functional is available, and the implementation procedure itself is not complex, which makes it possible to add into DFVM new activation functions if needed.

The implementation of sigmoid function in DFVM in C++ is represented as follows:

```
BaseFlow* DoubleFlow::sigmoid()
{
  return new DoubleFlow(1.0/
    (1.0+exp(-this->value)));
}
```

where `BaseFlow` is a superclass of flow, `DoubleFlow` is a class for *double* type, and `this->value` is an input value. This method returns new instance of `DoubleFlow`.

DoubleFlow.h file looks as follows:

```
#pragma once
#include "BaseFlow.h"
#include "DFVMExceptions.h"
#include "IntegerFlow.h"
#include "StringFlow.h"
#include "BooleanFlow.h"
#include <iostream>

class DoubleFlow :public BaseFlow
{
public:
    explicit DoubleFlow(const double val);
    virtual ~DoubleFlow() {}
    void print() const override;
    BaseFlow::Type type() const override;
    ...
    BaseFlow* sigmoid() override;
private:
    double value;
};
```

V. Research Results And Future Investigations

An advantage of the neural network description as data flows at the virtual machine is that such an approach does not require writing a program in any high-level language or such object-oriented programming languages as C++ or Python. In a case of the data flow virtual machine, the neural network description is just presented in a declarative form while calculations are being provided by a specialized virtual machine written in C++.

In the next implementations of DFVM a gradient descent method will be implemented to train neural networks using data flow based description, which will allow training neural networks in the same data flow virtual machine without additional libraries.

A disadvantage of the approach mentioned is that it is required to describe connections with each neuron in the layers. However, if to provide support for matrices in the virtual machine, then it would be possible to describe each layer as a single entity in a JSON file, which will make it possible to present complex large-scale neural networks in some lines.

For high-dimensional neural networks, DFVM will support an array data type for *double* and corresponding description for appropriate operations, such as multiplication, addition, etc. This will allow to describe connection of a data flow with massive data and to lessen number of lines in a JSON file. The next version of DFVM will be presented for bigger examples of practical neural networks with connection of layers as data flows and proper mathematical training methods.

References

[1] K. V. Kharchenko, “Extension of the LLVM virtual machine with parallel instructions to implement a message transfer system,” 2012 System analysis and information technology 14th Int. Conf., Kyiv, Ukraine, p. 302, 24 April 2012.

[2] K. V. Kharchenko, “Dataflow control paradigm and dataflow graphic presentation in SOA,” East-European journal for advanced technologies, no. 3/9 (69), pp. 22-29, 2014.

[3] K. V. Kharchenko, “An Architecture and Test Implementation of Data Flow Virtual Machine,” 2016 System analysis and information technology 18th Int. Conf., Kyiv, Ukraine, p. 268, 30 May – 2 June 2016.

[4] K. Kharchenko, O. Beznosyk and V. Romanov, “A Set of Instructions for Data Flow Virtual Machine,” IEEE First Ukraine Conference on Electrical and Computer Engineering (UKRCON 2017), Kyiv, Ukraine, pp. 931-934, 29 May – 2 June 2017.

[5] B. Lu, B. L. Evans and D. V. Tosic, "Simulation and Synthesis of Artificial Neural Networks Using Dataflow Models in Ptolemy," 4th Seminar on Neural Network Applications in Electrical Engineering NEUREL-97, Belgrade, Serbia, pp. 84-89, Sep. 8-9, 1997.

[6] M. Bacis, G. Natale, E. Del Sozzo and M. D. Santambrogio, “A pipelined and scalable dataflow implementation of convolutional neural networks on FPGA,” 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lake Buena Vista, FL, pp. 90-97, 2017.

[7] Y. H. Chen, J. Emer and V. Sze, "Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators," in IEEE Micro, vol. 37, no. 3, pp. 12-21, 2017.

[8] Jeffrey Dean et al. (2015, November 9). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[9] Theano GitHub [Online]. Available: https://github.com/Theano

[10] MXNet: A Scalable Deep Learning Framework [Online]. Available: https://mxnet.apache.org/

[11] Microsoft Cognitive Toolkit [Online]. Available: https://www.microsoft.com/en-us/cognitive-toolkit/

[12] Keras Documentation [Online]. Available: https://keras.io/

[13] Torch GitHub [Online]. Available: https://github.com/torch/torch7

[14] Torch. Scientific computing for LuaJIT [Online]. Available: http://torch.ch/

[15] Caffee Deep Learning Framework [Online]. Available: http://caffe.berkeleyvision.org/