IMPLEMENTATION SOLUTIONS FOR DEEP LEARNING NEURAL NETWORKS TARGETING VARIOUS APPLICATION FIELDS

Dana-Mihaela Petroşanu [1*]
Alexandru Pîrjan [2]

ABSTRACT

In this paper, we tackle important topics related to deep learning neural networks and their undisputed usefulness in solving a great variety of applications ranging from image and voice recognition to business related fields where they have the potential to bring significant financial benefits. The implementations and scale sizes of deep learning neural networks are influenced by the requirements of the developed artificial intelligence (AI) applications. We are focusing our research on certain application fields that are most suitable to benefit from the deep learning optimized implementations. We have analyzed and compared the most popular deep learning libraries available today. Of particular interest was to identify and analyze specific features that must be taken into account for in accordance with the tasks that have to be solved.

KEYWORDS: *Deep Learning, Artificial Intelligence, Deep Learning Libraries.*

1. INTRODUCTION

The concept of deep learning has existed for a long period of time, being called in different ways, according to different perspectives and moments in time. In the last years, the amount of data has increased significantly, more and more data being possible to be used in the training process of artificial neural networks (ANNs), thus increasing the usefulness and applicability of the deep learning concept. The evolution of the parallel hardware architectures has made it possible to develop complex deep learning structures having huge numbers of processing elements (neurons). Therefore, deep learning models have evolved along with the computational resources, becoming able to tackle successfully complex problems with a high level of accuracy.

According to the scientific literature [1], the concept of "deep learning" originated in the 1940s and one can identify three stages in its evolution. In the first stage, during the years 1940s and 1960s, the concept was called "cybernetics" and is marked by the development of the perceptron concept, that made it possible to train solely a neuron. Afterwards, in the period 1980-1995, the concept was called "connectionism", being strongly influenced by the development of the back-propagation technique that enabled the possibility to train

[1]* corresponding author, PhD Lecturer Department of Mathematics-Informatics, University Politehnica of Bucharest, 313, Splaiul Independentei, district 6, code 060042, Bucharest, Romania, danap@mathem.pub.ro
[2] PhD Hab. Associate Professor Faculty of Computer Science for Business Management, Romanian-American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania, alex@pirjan.com

ANNs having at most two hidden layers. The concept of "deep learning" was coined in 2006, during the third stage [1].

According to [2], the concept of deep learning can be viewed as a family of machine learning techniques, that exhibit certain common characteristics. Thus, they consist in several layers of neurons, interlinked in order to identify, extract and process certain characteristics. The output produced by each intermediary layer is used as an input for the following one. These techniques can employ either supervised or unsupervised methods. Common features are identified and are being processed in order to obtain specialized characteristics, arranged according to their rank. The techniques can achieve more levels associated with different concepts, related to the abstraction degree.

Interlinking is the main characteristic on which the deep learning concept relies upon. The whole principle of deep learning lies in the fact that even if a single element (also called a neuron) does not exhibit intelligent features, several elements interlinked together might have the capacity to unveil such qualities. Therefore, a key factor in the context of achieving an intelligent behavior consists in the number of processing elements that should be extensive [1].

Over the years, it has been proven that the networks' sizes significantly influence the accuracy of the obtained results and make it possible to approach complex tasks. If we are to compare the sizes of today's neural networks with the ones from forty years ago, we would notice that the dimensions of these networks have increased dramatically. In order to be able to implement huge sized networks, one must have access to significant computational hardware and software resources.

In the following section, we analyze the most important issues regarding the implementation of deep learning neural networks.

2. IMPLEMENTATION ASPECTS OF DEEP LEARNING NEURAL NETWORKS

When training deep learning neural networks one can identify several approaches. A classical approach consists in training the networks, using the Central Processing Unit (CPU) of only one computer. Nowadays, this way of dealing with the problem does not provide sufficient computational resources. The modern approach consists in using multiple processing nodes distributed over an entire network and modern parallel processing architectures such as those of the graphics processing units (GPUs).

The neural networks have high requirements from the computational point of view and thus one has to overcome the serious limitations of classical CPUs. There are several optimization techniques that one can apply in order to improve the performance of central processing units and benefit at maximum from the architectural features such as: the multi-threading mechanism; aligning and padding the data as to allow the central processing unit to retrieve the data optimally; enclosing supplementary data bytes in-between the existing data structures; devising appropriate floating or fixed type execution plans; reducing at minimum the expending of memory by sorting in a descending order according to the width of the elements; devising customized numerical computational procedures [1].

A lot of people who carry out scientific activities related to machine learning tend to overlook the above-mentioned implementation aspects thus risking to become limited in terms of the maximum number of neurons and of the obtained accuracy. Nowadays GPUs have become the platform of choice when developing state-of-the-art implementations of artificial neural networks. Initially, graphics processing units targeted exclusively the high computational requirements of video gaming applications [3]. The massive processing power of the GPUs offers great advantages in optimizing applications from various fields [4] and thus in the development of neural networks.

The type of processing that the GPU performs is relatively straight forward when compared to the central processing unit's tasks that necessitate frequent shifts of the execution to different sequences of instructions. One can parallelize without difficulty most of the processing tasks as the vast majority of the required computations are not depending on each other. When designing the graphics processing units, the manufacturers had to take into account the necessity of obtaining a large amount of parallelism and a high transmission capacity of the data fetched from memory, being compelled to reduce the GPU clock frequency and its ability to shift frequently the execution to different sequences of instructions. In order to develop an artificial neural network, the scientists must use various buffer settings in order to store the network's parameters, the threshold value at which the activation occurs, taking into account the fact that all these values must be computed for each new step of the training process [1].

The graphics processing units represent a better solution in implementing deep learning neural networks, as they offer a higher transmission capacity of the data fetched from memory, when compared to the one of the central processing units. Furthermore, as in the development process of an artificial neural network (ANN) there are not required frequent shifts of the execution to different sequences of instructions, the graphics processing units can successfully manage the whole execution process, being best suited for this kind of tasks. The software operations related to the ANNs have the potential to be decomposed in multiple tasks, that can be managed in parallel, by neurons belonging to the same layer. Therefore, these operations could be carried out easily as one employs the parallel processing power of the graphics processing units.

Initially, when the graphics processing units were developed for the first time, their architecture was restricted exclusively to graphics processing purposes. As the time passed, the graphics processing units have evolved, offering a greater flexibility towards specialized functions that could be called for different allocation and transforming operations. What was interesting is that, for the first time, these operations needn't have to be solely for graphics processing. Consequently, the graphics processing units have become a viable tool for performing scientific computations by using its resources that were initially developed only for graphics rendering.

After the Nvidia company has introduced the Compute Unified Device Architecture (CUDA), the graphics processing units have evolved and become general-purpose graphics processing units that were able to execute source-code and not only specialized functions. In light of the new possibilities that emerged due to the huge parallel processing power and increased memory bandwidth, the graphics processing units that

incorporated the CUDA architecture became the platform of choice for scientists that were prospecting the field of deep learning [1].

In order to harness the full computational potential of a graphics processing unit, the developers must apply optimization techniques other than those used for central processing units. In contrast with central processing units, that use the cache mechanism to improve the overall performance of a source code, the graphics processing units might perform several operations more times in parallel and obtain an improved performance when compared to the CPU.

For a programmer to achieve top peak performance on a graphics processing unit, he must make full use of the Compute Unified Device Architecture threading mechanism and carefully manage the threads within blocks of threads and the blocks in grids of blocks of threads. Particular attention must be payed to managing correctly and efficiently the memory. One must take into account techniques such as memory coalescing but most of all the features offered by the respective CUDA architecture in order to improve the software performance of a neural network. An important implementation aspect when developing a neural network using a CUDA GPU consists in assuring that every thread of a group of threads processes the same task, in the same time, in parallel.

The graphics processing unit is not suited for frequent shifts of the execution to different sequences of instructions and this is the reason why one must not completely neglect the central processing unit, the best performance being obtained when using a hybrid approach for developing artificial neural networks.

In CUDA, the execution threads within a block of threads are grouped in warps, containing 32 threads. This dimension represents the smallest amount of data that is processed by a Compute Unified Device Architecture multiprocessor, according to the "Single Instruction, Multiple Data" category of the Flynn's taxonomy [5]. Therefore, during an execution cycle, the threads within the same warp process a single instruction. If different instructions should be processed by the threads within the same warp, they are executed sequentially.

Sometimes, in their research, the developers need to check the quality, performance or reliability of new algorithms or models and in this purpose, they often use software libraries, developed in various programming languages, containing high performance software packages, useful for developing their applications. For example, in the Machine Learning field, the libraries are developed in Python, Java, .NET, C, C++, Lua and other programming languages [1], [6], [7]. Some of the most popular deep learning libraries are categorized by their programming language and synthetized in **Table 1**.

Table 1. Some of the most popular deep learning libraries

No.	The programming language	Deep learning libraries/toolboxes developed in the respective programming language
1	Python	Theano and based on it: Keras, Pylear2, Lasagne, Blocks
		Caffe
		nolearn
		Gensim
		Chainer
		deepnet
		Hebel
		CXXNET
		DeepPy
		DeepLearning
		Neon
2	Python API over a C/C++ engine	TensorFlow by Google
3	Matlab	ConvNet
		DeepLearnToolBox
		cuda-convnet
		MatConvNet
4	C++	eblearn
		SINGA
		NVIDIA DIGITS
		Intel® Deep Learning Framework
		Microsoft Cognitive Toolkit, previously known as CNTK
5	Java	N-Dimensional Arrays for Java (ND4J)
		Deeplearning4j
		Encog
		H20 Web API
6	JavaScript	Convnet.js
7	Lua	Torch
8	Julia	Mocha
9	Lisp	Lush (Lisp Universal Shell)
10	Kaskell	DNNGraph
11	.NET	Accord.NET
12	R	darch
		deepnet

The Python programming language was developed by Guido van Rossum and launched in 1991 as a GPP (general-purpose programming) high-level language. There are numerous frameworks and libraries developed in Python. For example, the Theano library can be used for processing mathematical operations. This library facilitates the development of

deep learning algorithms. Theano is the foundation for several other libraries that are based on it: Keras, Pylear2, Lasagne, Blocks [8].

Keras represents a deep learning neural network library, comprising several modules, that can be successfully used for processing tensors using either the graphics processing unit or the central processing one. The Pylear2 library comprises a large set of algorithms that can be successfully used in the development of deep learning neural networks. The Lasagne library has been developed in modules, being characterized by easiness, clearness, having in mind the success of its practical applications. Another developing framework useful in developing deep learning libraries, based on the Theano library, is Blocks [9].

One of the most popular framework for developing deep learning neural networks is Caffe, developed in Python by BVLC (Berkeley Vision and Learning Center) and other developers of the software community. The primary characteristics based on which this framework has been designed, consist in an efficient processing, modularization in order to obtain a state-of-art framework. Using the Caffe framework, Google has developed its DeepDream project, a library written in the C++ programming language, licensed under the Berkeley Software Distribution (BSD), having a Python interface [6].

Another deep learning library developed in Python is nolearn, that comprises a set of machine learning tools and makes use of wrapping and abstraction, targeting already developed libraries [10]. Gensim is a specially developed toolkit in Python, that exposes several high-performance algorithms, designed for developing deep learning neural networks that have to process huge amounts of text data [8].

Chainer, programmed in Python, facilitates the implementation of deep learning neural networks, offering support for certain algorithms. One prominent characteristic of Chainer is its flexibility, being easy to use and understand [9]. Another deep learning implementation achieved using the Python language is deepnet, an implementation that is based on Graphics Processing Units. Many of the most well-known deep learning algorithms have been implemented within deepnet on the GPUs in order to facilitate the development of deep learning neural networks [6].

The Hebel library was developed to facilitate the development of deep learning neural networks in the Python programming language, using PyCUDA that makes it possible to benefit from the huge parallel computational power of Graphics Processing Units that incorporate the Compute Unified Device Architecture (CUDA). It offers support for the most well-known and efficient kinds of neural networks [10].

CXXNET is a reliable deep learning framework where the processing is spread among multiple processing nodes. It offers support for the CUDA-C language and incorporates user-friendly interfaces useful for developing the networks [8]. DeepPy is a framework for deep learning, under the free software license of the Massachusetts Institute of Technology (MIT), being based on the Python programming language. The existing source code can be extended with ease and also offers support for Nvidia Graphics Processing Units that incorporate CUDA technology [9].

DeepLearnig is a library, that was programmed in C++ and Python and contains high performance software packages, useful for developing deep learning neural networks [6].

Based on the Intel Nervana Python comes Neon, a framework for creating deep learning neural networks, having as a main declared goal to attain the best in-class results [10].

The TensorFlow software library is written with a Python API over a C++ engine and was developed by the Google researchers. This library is useful in carrying out the research related to deep learning neural networks and many other scientific fields. TensorFlow is designed to solve problems related to numerical computations through graphs. Within such a data flow graph, the nodes are the mathematical operations and the edges are the tensors. The library can be easily implemented on several central processing units or graphics processing units in different environments through the same Application Programming Interface (API) [10].

The popular development environment Matlab offers many useful software instruments for developing and implementing a large variety of neural networks: ConvNet, DeepLearnToolBox, cuda-convnet, MatConvNet. Convolutional neural networks (ConvNet) represent powerful deep learning tools for classifying different elements, by acquiring knowledge on their own from unprocessed data.

Another software instrument for developing deep learning networks is the DeepLearnToolBox that is not active anymore, being considered obsolete. Still, it contains different types of deep learning neural networks like deep belief networks and convolutional networks along with different autoencoder types [10]. Cuda-convnet is another useful implementation of the feed-forward artificial neural networks developed in Matlab. It is implemented in C++/CUDA, being a high-performance toolbox. Its training is based on the backpropagation algorithm.

Another Matlab toolbox is MatConvNet that implements Convolutional Neural Networks (CNNs) and is useful in applications that require the automated extraction of information from images. The main properties of this toolbox are its simplicity, efficiency and capacity of running the most important Convolutional Neural Networks, among which it is worth mentioning applications for image or text detection, sorting, segmenting, recognition [6].

There are a lot of deep learning frameworks available that use the C++ programming language as their basis, for example: eblearn, SINGA, NVIDIA DIGITS, Intel Deep Learning Framework and Microsoft Cognitive Toolkit, previously known as CNTK. Eblearn represents a useful C++ library, under an open-source license, developed at the New York University. It is useful in developing different types of CNNs, providing a friendly Graphical User Interface [10].

Another deep learning framework is SINGA, developed in 2014 in Singapore, at the National University. SINGA is supported by an American non-profit corporation, ASF (Apache Software Foundation). This library is based on decomposing data between the nodes of a cluster and employs a parallel training process. It is useful in managing the interactions between computers and human languages (natural language processing) and supports many deep learning classes of models [6].

Developed by Nvidia, DIGITS (Deep Learning GPU Training System) represents a powerful tool, useful in training in a reduced time DNNs (deep neural networks) that are capable to classify images and detect objects. It has as a main advantage the fact that is

interactive and therefore the researchers can be preoccupied mainly by developing the neural networks, without having to be concerned about issues regarding writing computer programs, identifying and removing errors.

Intel Deep Learning Framework (IDLF) represents a SDK (software development kit) library, providing a software framework that fuses together the Intel platforms and is useful in training, executing and accelerating DNNs. The main characteristics of IDLF are: it offers support for a rich variety of accelerators; its development is based on an optimized code; it supports the development of a wide range of ANNs on the same platform; it is suitable for cloud computing by devising schemes for allocating the tasks among different processing nodes; it enables the improvement of the training process during its execution [6].

Microsoft Cognitive Toolkit, previously known as CNTK is a deep learning library developed in C++ that offers high accuracy and speed, being compatible with many common programming languages or algorithms. This toolkit is characterized by a series of capabilities and features that it offers to the users. It contains components that are able to manage sparse or dense data from other programming languages, being suitable for both unsupervised and supervised learning. It also contains components that are able to handle massive datasets. Microsoft Cognitive Toolkit is characterized by an efficient use of resources, offering a high level of parallelism on multiple processing units and an optimized mechanism of memory sharing, useful in managing large models in the memory of graphics processing units. The toolkit offers a full application programming interface useful in developing neural networks, evaluating models, ensuring suppleness and flexibility [10].

A series of deep-learning libraries are developed in the Java general-purpose programming language, for example: N-Dimensional Arrays for Java (ND4J), Deeplearning4j, Encog and H20 Web API. The first of these libraries, ND4J, is designed for the Java virtual machines (JVMs), an abstract machine for automatically performing computations, that makes it possible for a computer to run Java software instructions. ND4J is characterized by the fact that it runs fast specific routines, requiring a small amount of random access memory [11].

The Deeplearning4j deep-learning library is written for Java and Scala programming languages, under an open-source license, developed mainly in order to be used in managing the external and internal factors that affect the functions of companies, its usefulness in research having lesser extent. Encog is another software instrument that has been evolving since 2008, being useful in developing deep learning networks, supporting a wide range of learning algorithms and neural networks. It is useful in many scientific fields, especially in medicine and finance [11].

H2O is another deep-learning library developed in Java, under an open-source license. It is able to scale to more processing nodes, offering a high-level of performance and access to complex algorithms that enable programmers to develop powerful applications, using an intuitive application programming interface. A lot of companies have developed complex expert systems that help improve their economic activity. H2O is able to store and process billions of tuples in-memory using a specialized compression algorithm. H2O

offers access to familiar application programming interfaces and also an incorporated web interface [6].

Convnet.js represents a deep learning library developed in Javascript, having as a unique characteristic the ability to develop deep learning neural networks using solely an internet browser. Convnet.js has the same software and hardware requirements as the browser, being able to train and implement ANNs without a hassle. It can also be deployed using the supplied Javascript source code file "node.js" in a server [10].

Another programming language useful in developing deep learning neural networks libraries is Lua, developed in 1993. Although being used extensively in the videogames industry, Lua has been used successfully in developing a wide range of popular commercial applications, proving to offer a high-level of performance, efficiency, ease of programming, a little consumption of resources.

Based on Lua, in 2002 was released Torch, a consistent framework that provides access to algorithms optimized for the Compute Unified Device Architecture enabled GPUs, useful in the machine learning field. The main aim of the framework Torch is to offer the greatest extent of adaptability, reduced time in developing specialized algorithms with minimum effort. There are a lot of popular social networks, search engines, universities and research institutes that use Torch in their everyday activities [12].

A programming language designed for numerical computations is Julia, released in 2012. Julia includes a complex compiler along with a comprehensive specialized mathematical library, offering support for executing the tasks in parallel on multiple processing nodes. This programming language was used in developing Mocha, a specialized framework for deep learning neural networks, being influenced as a development model by the above analyzed Caffe framework. Mocha implements specialized numerical solvers and tools useful in training CNNs. The most important characteristics of Mocha are represented by its modularity, complex interface, easiness in portability, being compatible with multiple JavaScript assertion libraries [13].

Lisp represents an ensemble of programming languages, that dates back to 1958. It was released one year after the Fortran programming language and it is still widely used today. Initially Lisp was developed as a way of facilitating the mathematical notations in the software programs and soon afterwards it began to be the language of choice for the scientists in the artificial intelligence field. Lisp introduced for the first time many programming paradigms. Based on Lisp, it was developed under the General Public License the Lush object-oriented programming language targeting the scientific field. It has a wide range of applications (machine learning, image and signal processing, extracting knowledge from data, etc.) and can overcome the limitations of other consecrated development environments [6].

The Programming language Haskell was released in 1990, targeting diverse applications' domains without accepting to change the state of an object or to modify it after it has been created. There are various implementations for Haskell released under the open source license, some of them being compliant with the Haskell 98 standard ("Glasgow Haskell Compiler", "Utrecht Haskell Compiler", "Jhc", "LHC" etc.) and other implementations are not kept active or in maintenance anymore.

In Haskell, it was developed DNNGraph, a DNN domain specific language for developing the network's structure. DNNGraph makes use of the different libraries like the "lens library" and a graph oriented library "fgl" for defining the structure of the network and also a series of optimization strategies. DNNGraph is able to generate files compatible with the above analyzed Caffe and Torch frameworks [10].

In the extremely popular .NET framework it was developed Accord.NET that offers artificial intelligence capabilities along with specific software libraries in the fields of sound and graphics processing. Accord.NET offers the necessary tools to build commercial applications. It offers to the developer a lot of ready-made templates and the possibility to exchange and interchange the machine learning algorithms with ease [6].

A very popular programming language, released in 1993 under an open source General Public License is R, a development environment that facilitates computation in the field of statistics and image processing. R offers a CLI (command line interface) and there are also available a few graphical user interfaces [10].

There are several frameworks and tools developed in the R programming language designed for building DNNs, for example darch package and deepnet. The darch package offers the possibility to train the neural networks in advance using the "contrastive divergence method" and popular specialized algorithms that make detailed adjustments to the networks' parameters. Another framework, developed based on the R programming language is deepnet, that offers several DNNs architectures, specialized algorithms and encoders [6].

In the next section, we present a series of strategies, useful in improving the software performance of the deep learning neural networks implementations.

3. STRATEGIES FOR IMPROVING THE PERFORMANCE OF DEEP LEARNING NEURAL NETWORKS IMPLEMENTATIONS

Deep learning neural networks can be successfully implemented in image, video, sound, text recognition or processing and in obtaining accurate predictions in various scientific fields, such as economy, mathematics, physics, neuroscience, medicine, pharmaceutical industry. Most of the electronic means of payments and micropayments need solutions that can identify, prevent and counteract frauds [14], [15]. Deep learning neural networks offer new possibilities to secure the electronic means of payments, being able to identify fraud faster and more accurately than the human factor. Used along with other algorithms, deep learning represents a powerful tool for classifying, clustering and forecasting, based on the input data (**Figure 1**).

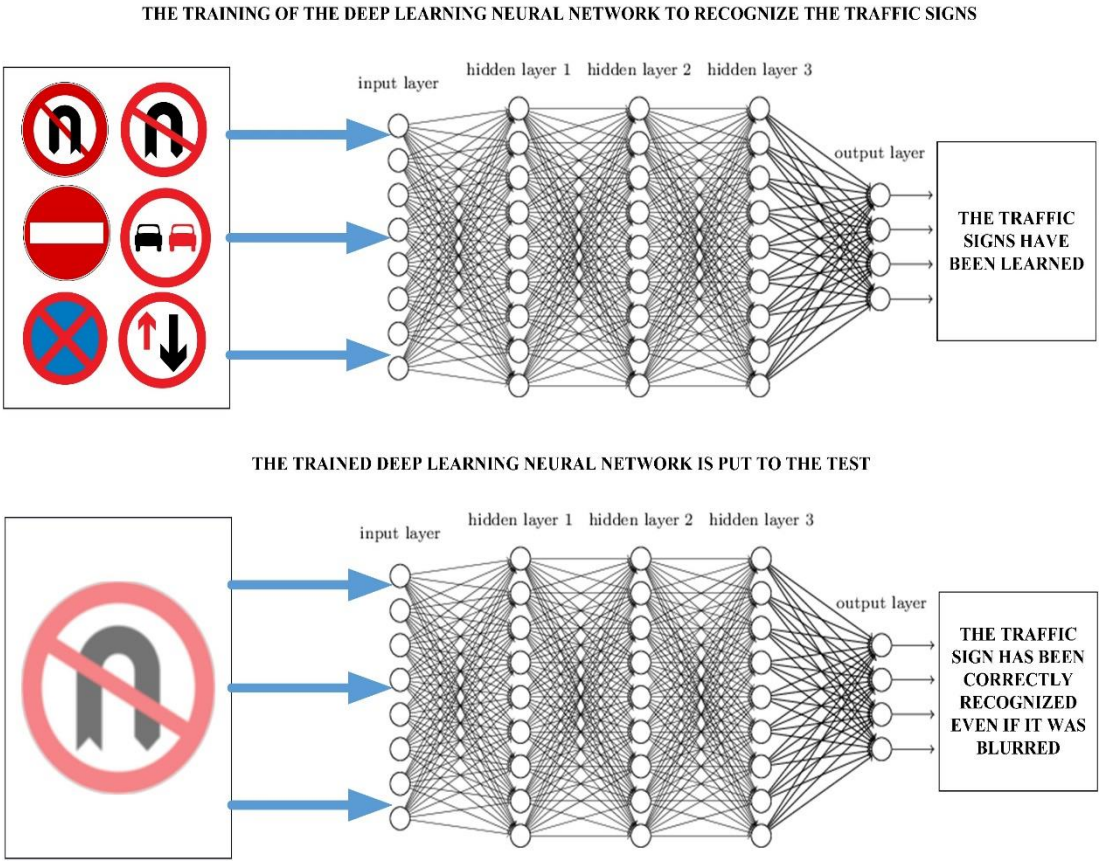

Figure 1. A deep learning neural network that recognizes the traffic signs[1]

In **Figure 1** it is represented a deep learning neural network that recognizes the traffic signs. The neural network is trained using as an input a set of traffic signs. After the network has been trained, it is put to the test using as an input a blurred traffic sign, as if it were affected by poor meteorological traffic conditions when driving.

When developing large neural networks that have to process a lot of data in order to obtain the results, a huge computing power is required. A single workstation is not sufficient for this kind of computational volume. As a consequence, a distributed system comprising multiple processing nodes (workstations) is an appropriate solution for training and achieving the results in due time. The processing in parallel of data is easily achieved in the case of deep learning neural networks as every input set that has to be inferenced can be executed by a different node. Another method for attainting data parallelism is to divide the data in several parts and allocate each part to more processing nodes [1].

[1] The figure has been created using the software tool Visio 2016, by inserting "Online Pictures" type elements, tagged with reusable "Creative Commons Licenses".

One must note that it is much more difficult to process the data in parallel throughout the training phase. For example, in the case of the scaled gradient descent one would obtain better results if he allocates the steps that have to be computed to several processing nodes of the distributed system. However, this approach cannot be achieved as the values obtained at a certain step depend on the values obtained at a previous one.

In the scientific literature [1], [16], [17], it is proposed an asynchronous approach that consists in allocating portions of the memory where the values reside among multiple processing cores. Every variable is unlocked during the processing in order to assure a concurrent access of the processing cores to the values. Nevertheless, this solution has the drawback of diminishing the enhancement that should be obtained when progressing to a new step of the algorithm as several processing cores can sometimes store new data over the existing contents of a variable and thus cancel the progress up to that moment. Although, an undisputed advantage of the asynchronous approach consists in the speedup of the whole learning phase.

Other variations of this method are proposed in the literature that consist in handling the values using a dedicated server [18]. The asynchronous approach implemented on a distributed system yields notable results being at the forefront of the training process for deep learning neural networks.

The most important aspect in off-the-shelf software is to reduce as much as possible the execution time and the memory load when computing the results rather than when training the neural network. It is not uncommon for a certain neural network to be trained using high computational resources and afterwards to be implemented and put into use in an environment where the hardware resources are more consumer-oriented.

In order to evaluate a specific developed model, it is often used a model compression strategy, within which an initial model is replaced with another one, having a smaller size, which requires a reduced amount of memory and offers the benefit of a reduced execution time. This technique is suitable for the cases when the initial model has a big size. In this case, several smaller models are designed and tested, finally replacing the initial one with the set of models, thus obtaining the model with the smallest error of generalization. Analyzing and assessing all the developed models could become an intense resource consuming task [19].

In some situations, using only one model can yield better results if its size is large enough. In the cases when the number of the existing training elements is reduced, one must use a larger number of parameters than are required by the specific problem. After training the neural network, one can simply obtain a new larger training set of elements by using it. Afterwards, using these elements, one can train other models, having reduced sizes, that offer very good results using as training sets, different subsets of the new larger training set. The training data must be eloquently sampled so that the neural network can provide correct results when it is being applied in a real word scenario [1].

Among the strategies for improving the performance of data processing systems, one of the most important strategy consists in implementing a dynamic structure regarding the graph that reflects the necessary computations for processing the set of input data. In the deep neural network's case, the easiest way of implementing the dynamic structure

approach consists in allocating properly the group of machine learning models that are suitable for a certain set of input data [20].

Another important method, useful in improving the performance and reducing the execution time for a data processing algorithm that implements the classification task, consists in training and using a sequence of such algorithms (classifiers), thus obtaining a cascade approach. This strategy is suitable for the cases when one aims to identify with high accuracy the occurrence of rare objects or events.

The Google search engine's researchers have implemented the cascade approach in many situations, for example when one wants to transcribe the numbers of the addresses that have been identified using the Street View technology. They have used an authentication method that comprises two steps. In the first step, the process detects the location of the address number using a specific machine learning model. Afterwards, in the second step, another model is used in order to transcribe the desired number [1].

Due to the undisputable advantages and usefulness of the deep learning neural networks, the researchers are concerned about discovering and implementing more effective strategies for improving the performance of these networks. This field of research represents an open topic and a permanent challenge for the scientists, information technology (IT) specialists, mathematicians, engineers and economists worldwide.

4. CONCLUSIONS

Lately, the deep learning class of machine learning algorithms become more and more popular among researchers in various fields, such as speech, audio, graphics or pattern recognition, processing of natural language and bioinformatics. Also, a wide range of architectures relying on this concept have emerged, like deep, convolutional deep, deep belief, recurrent artificial neural networks.

In our paper, we have first introduced the main concepts related to the deep learning neural networks and their state of art from the literature. Afterwards, we have analyzed implementation aspects of deep learning neural networks, we have revealed and justified their undisputed usefulness in solving a great variety of applications, highlighting the fact that the requirements of the developed applications influence the implementations and scale sizes of deep learning neural networks. We have paid a special attention to the analysis and comparison of the most popular deep learning libraries/toolboxes available today and the programming language in which they were developed. We have also highlighted the most important strategies for improving the performance of deep learning neural networks implementations.

Taking into account the usefulness of deep learning neural networks, the benefits that they offer in various research fields, in industry, in economy, in IT and games industry, the possibility of implementing these networks in the GPUs and employ their huge parallel computational power, we can conclude that the deep learning neural networks represent a functional, practical and efficient solution for successfully achieving outstanding results in a wide class of domains.

REFERENCES

[1] Goodfellow I., Bengio Y., Courville A., *Deep Learning (Adaptive Computation and Machine Learning series)*, Publisher: The MIT Press, 2016, ISBN-10: 0262035618, ISBN-13: 978-0262035613.

[2] Deng L., Yu D., *Deep Learning: Methods and Applications (PDF)*. Foundations and Trends in Signal Processing. 7 (3–4), pp. 197-387, 2013, DOI: 10.1561/2000000039.

[3] Lungu I., Pîrjan A., Petroşanu D. M., *Solutions for Optimizing the Data Parallel Prefix Sum Algorithm Using the Compute Unified Device Architecture*, Journal of Information Systems & Operations Management, Vol. 5, Nr. 2.1/2011, pp. 465-477, ISSN 1843-4711.

[4] Petroşanu D. M., Pîrjan A., *Economic considerations regarding the opportunity of optimizing data processing using graphics processing units*, JISOM, Vol. 6, Nr. 1/2012, pp. 204-215, ISSN 1843-4711.

[5] Padua D., *Encyclopedia of Parallel Computing*, Springer Publishing Company, Incorporated, 2011, ISBN:0387097651 9780387097657, pp 689-697.

[6] Clarke D., Daoud's Page on Github, *17 Great Machine Learning Libraries*, http://daoudclarke.github.io/machine%20learning%20in%20practice/2013/10/08/machine-learning-libraries, accessed on March 22, 2017.

[7] Tăbuşcă A., *Learning a programming language for today*, Journal of Information Systems & Operations Management, Vol.9, No.1/2015, pp. 83-94, ISSN 1843-4711.

[8] Matthes E., *Python Crash Course: A Hands-On, Project-Based Introduction to Programming*, No Starch Press, 2015, ISBN-10: 1593276036, ISBN-13: 978-1593276034.

[9] Raschka S., *Python Machine Learning*, Packt Publishing ISBN-10: 1783555130, ISBN-13: 978-1783555130, 2015.

[10] Teglor, http://www.teglor.com/b/deep-learning-libraries-language-cm569/, accessed on March 22, 2017.

[11] Kaluza B., Machine Learning in Java, Packt Publishing, 2016, ISBN-10: 1784396583, ISBN-13: 978-1784396589

[12] Ierusalimschy R., Programming in Lua, Fourth Edition, Publisher: Lua.Org, 2016, ISBN-10: 8590379868, ISBN-13: 978-8590379867

[13] Russel S., Sengupta A., Hanson L., *Learning Julia: Rapid Technical Computing and Data Analysis*, O'Reilly Media, 2017, ISBN-10: 1491903600, ISBN-13: 978-1491903605

[14] Pîrjan A., Petroşanu D. M., *Dematerialized Monies – New Means of Payment*, Romanian Economic and Business Review, Vol. 3 Nr. 2/2008, pp. 37-48, ISSN 1842–2497.

[15] Pîrjan A., Petroşanu D. M., *A Comparison of the Most Popular Electronic Micropayment Systems*, Romanian Economic and Business Review, Vol. 3, Nr. 4/2008, pp. 97-110, ISSN 1842–2497.

[16] Bengio Y., Ducharme R., Vincent P., *A neural probabilistic language model*, in Advances in Neural Information Processing Systems 13 (NIPS'00), pp. 932–938, MIT Press, 2001.

[17] Recht B., Ré C., Wright S.J., Niu F., Hogwild: *A lock-free approach to parallelizing stochastic gradient descent*, Advances in neural information processing systems 24 (NIPS 2011), Curran Associates, Inc, Red Hook, NY, USA, pp. 693–701, 2011.

[18] Dean J., Corrado G., Monga R., et al., *Large scale distributed deep networks*, In Proceedings of Neural Information Processing Systems (NIPS), 2012.

[19] Bucilua C., Caruana R., Niculescu-Mizil A., *Model compression*. In: KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, ACM (2006) pp. 535–541, 2006.

[20] Bengio Y., *Deep learning of representations: Looking forward*, in Statistical Language and Speech Processing SLSP 2013, Lecture Notes in Computer Science, vol. 7978, Springer, Berlin, Heidelberg, pp. 1-37, 2013, DOI: 10.1007/978-3-642-39593-2_1.