Automatic Differentiation library for training Convolutional Neural Networks in Julia

1st Maciej Zalewski
Faculty of Electrical Engineering
Warsaw University of Technology
Warsaw, Poland
maciej.zalewski5.stud@pw.edu.pl

2nd Jan Janiszewski
Faculty of Electrical Engineering
Warsaw University of Technology
Warsaw, Poland
jan.janiszewski.stud@pw.edu.pl

I. Literature review

Convolution Neural Networks (CNNs) are powerful tools and are easily accessible nowadays. There are many papers and articles describing how CNNs work on a higher level, explaining architecture, each layer (convolution layer, pooling layer and fully-connected layer) and how these layers can by stacked [1].

Teaching the network, that is, increasing the accuracy of classification, requires counting many partial differentials. To perform this operation efficiently, most model implementations use automatic differentiation. The article [2] describes a Julia language library called Zygote which enables automatic differentiation in reverse-mode. The authors describe with what mechanisms they managed to create a flexible differentiation tool while maintaining optimal performance.

The article [3] provides an overview of libraries for automatic differentiation available in languages such as C++, F#, Python, Matlab and Julia. The libraries were compared in terms of their functionality and runtime, which ranged over three orders of magnitude in some tests.

Understanding how Automatic Differentiation works is crucial in building modern neural network. Proper implementation of AD with understanding which mode is better suited for the problem (forward or reverse) and memory management helps in algorithm optimization [4].

There already exist libraries implementing CNN. In programming language Python, most popular are Facebook's Torch (PyTorch) and Google's TensorFlow. One can find many internet tutorials but also papers regarding basic implementations of CNN based on TensorFlow [5] [6]. PyTorch utilizes Automatic Differentiation and GPU acceleration to speed up computations. While TensorFlow constructs static dataflow graph, PyTorch has define-by-run approach and builds graph dynamicaly, freeing unused parts and saving memory [7].

Julia is fairly new programming language compared to Python and is still behind when it comes to the popularity, number of available libraries and features. It stands out by balancing both efficiency and simplicity. Language is fast, easy to use and was designed for high-performance computing. Julia is still a developing language, low number of third-party packages can be a problem in less popular areas. On the other hand, it gives a large field for developers to create new solutions [8].

Flux.js, created by Mike Innes, is a popular machine learning library written in Julia which allows creating models and applies Automatic Differentiation. It is mostly used for natural language processing but can be successfully applied in other machine learning areas [9].

II. Purpose of the study

The purpose of the study is to compare the basic capabilities, that is speed and classification accuracy, of our library for automatic differentiation written in Julia with one of the existing and popular libraries. As a reference solution, we chose TensorFlow framework written in C++ and Python.

The first step was to build a library for automatic differentiation in Julia. Then we created a simple convolutional neural network using the new library, and finally we compared it with the reference solution using TensorFlow. For an effective comparison, the CNN network model is as simple as possible, multithreading and GPU acceleration are out of the scope of this study.

The reason for choosing Julia as the language in which the library was written was that it is a language created with speed and optimization in mind. At the same time, it is open source project on the MIT license, which supports mathematical operations in a convenient way (e.g. on matrices), without losing efficiency. This makes it an excellent candidate when it comes to a tool for creating systems using computationally complex algorithms operating on vectors and matrices. Such is the problem of creating a library for automatic differentiation, where the functions are called repeatedly for large amounts of data. Each unnecessary operation or memory allocation causes an an increase in computation time and memory consumption. With a single function call, this problem is negligible, but during network training, even a very simple one, the number of input objects reaches tens of thousands or even more. Adding epochs and images with many color channels as an input, we get a huge number of function calls and data to process, thus small errors in the form of additional allocations and operations sums up and significantly affect the final computation time.

Julia is fairly new language when compared with other languages in the field of machine learning. Number of available packages is lower than for Python and there is no support in many environments, like for example Google Colab. By creating simple automatic differentiation library in pure Julia we wanted to show, that it is possible to catch up with popular solutions given out of the box, while still having huge potential for further development in this field.

III. Study description

The model of convolutional neural network is as simple as possible to minimize the effect of network implementation and focus on the automatic differentiation, while still working properly in terms of recognizing images.

The convolutional neural network we implemented was compared with an analogous network but created in Python using the Tensorflow framework. The comparison was made by time-testing the execution of the entire training and testing procedure of the network and measuring classification accuracy.

A. Model description

The convolutional neural network model consists of:

- Convolution layer
- Max Pooling layer
- Flatten
- Fully Connected layer (Dense)
- Loss function

In convolution layer we have chosen to use only one filter. Kernel is of size (3x3) and stride is equal to 1. Activation function is ReLU, which is a popular choice for convolution layers because of it's simplicity and efficiency in computations.

For Max Pooling layer we used kernel of size (2x2), which allows to reduce the size of the data while maintaining enough information to train and test the network.

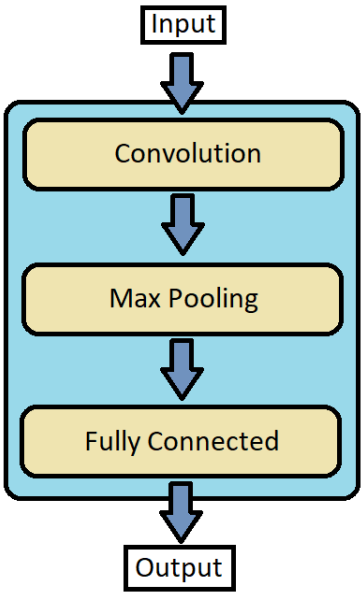
Normally after convolution and pooling most of the CNNs use another pair of these layers, but keeping in mind our aim of the study we omitted them. Accuracy of this model was acceptable for the purpose of comparing automatic differentiation libraries.

Before passing data to the next layer it must be flattened (matrix is transformed to vector form).

Last one is Fully Connected layer (also known as Dense), where activation function is softmax. This function is commonly used at the end to generate probabilities.

Data from the softmax function is used by the loss function to predict from which class the given input image is. For loss we have chosen Sparse Categorical Crossentropy.

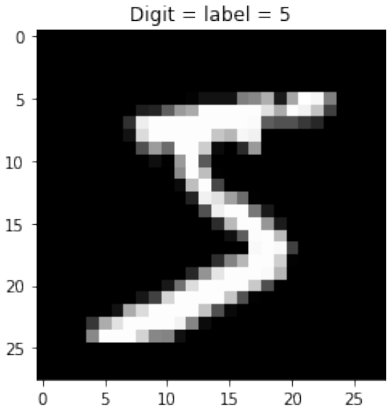
Additionaly, as the optimizer for updating weights during training we were using Gradient descent. While not being one of the bests, it's advantage is being simple. Learning rate was set to 0,001 and training took 3 epochs.



B. Dataset description

One of the most popular and commonly used dataset for training and testing image processing systems is MNIST. It is a large dataset of handwritten digits, containing 60,000 training images and 10,000 testing images.

Dataset is very well prepared, easy to download and use both in Julia and Python environments. Images are small, each one of them is 28x28 pixel size. They are in greyscale and pixels are coded in Float32, thus they were perfect choice for this study.



C. Julia project description

For project implementation Julia 1.8.5 was used. The following libraries were used:

- DrWatson 2.12.4 - manager for Julia oriented projects
- MLDatasets 0.7.9 - library with MNIST dataset that was used for testing networks

D. Python project description

For project implementation Python 3.10.5 was used. The following libraries were used:

- tensorflow 2.12.0 - framework for defining and running machine learning workloads. We used it to define simple convolutional neural network and train it.
- numpy 1.23.5 - library with utilities for working with multidimensional arrays, used for manipulating input data into right format

E. Description of the computing environment

All tests were performed on the same computing environment with the following characteristics:

- Operating system: Windows 11 Home
- CPU: AMD Ryzen 5 5600H 3.30 GHz
- RAM: 16 GB
- GPU: NVIDIA GeForce RTX 3050

F. Tests performed and timing methodology

Following tests were performed:

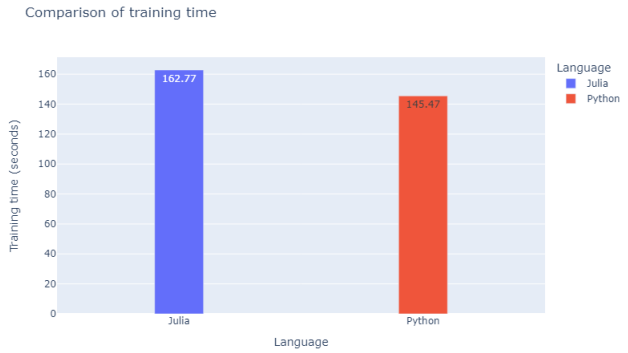
- Training time
- Testing time
- Classification Accuracy

Timing measurements were made through the Measure-Command tool, which allows the accurate measurement of any console command on Windows PCs. Using a universal measurement tool allows a fair comparison of the two solutions. Each test was repeated 10 times, the results presented in this article are averages of all repetitions.

IV. Test results

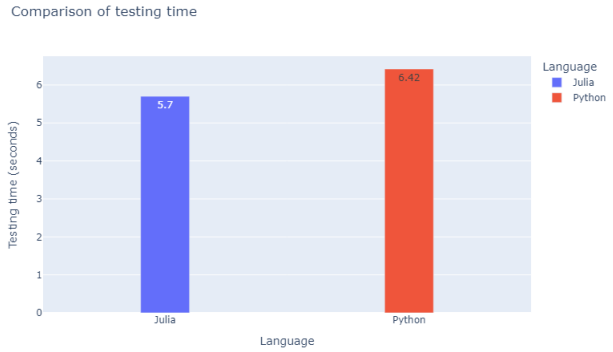
A. Training time

On below plot we can see comparison between training time of CNN created in Julia and in Python. Training Python network took 17 seconds less than Julia network.



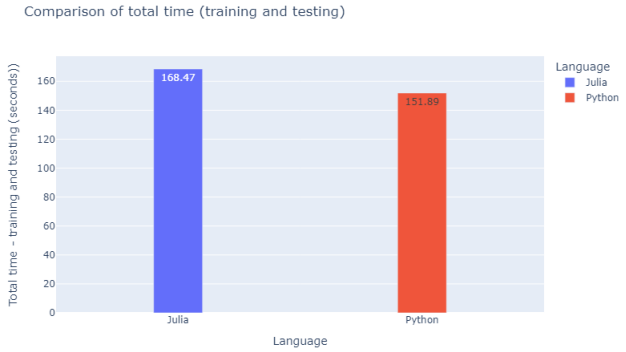
B. Testing time

On below plot we can see comparison between testing time of CNN created in Julia and in Python. Julia network was 0.7 seconds faster.



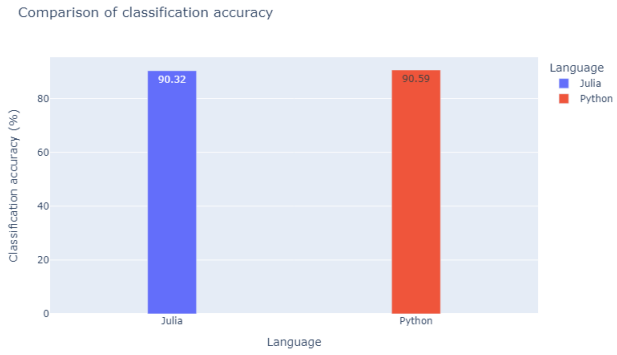
C. Total time

On below plot we can see comparison between total time of CNN created in Julia and in Python. Python network was overall $16,58$ seconds faster.



D. Accuracy

On below plot we can see comparison between accuracy of CNN created in Julia and in Python. Python network was $0,27\%$ more accurate. Taking into account the measurement error, both networks can be considered to be equally accurate.



V. Conclusions

A CNN created in Julia on the basis of our library for automatic differentiation achieved a classification accuracy equal to the reference solution in Python. Training the network we have created takes 17 seconds longer than with TensorFlow, this is not a relatively big difference but it cannot be ignored.

When it comes to testing i.e. making classifications by a trained CNN, time falls in favour of Julia. The difference is 0.7 seconds. Interestingly, the ratios between the times compared are almost the same. The Julia network trains 11% longer than the Python network. Python network takes 12% longer to classify than Julia network. When comparing the total time, the solution created using Python achieved a better result. This shows that there is room for further optimisation of the library we wrote, something that could particularly speed it up would be to minimise memory allocation.

As mentioned earlier Tensorflow, library used in Python referential example, although it allows neural networks to be created in the high-level language Python the most relevant parts of this library were written in C++. Working with a project based on at least two programming languages that are extremely different from each other can cause difficulties. Trying to customise the network or possibly solve bugs may force us to leave Python to look at and work with code written in C++. We also encountered this problem when we were trying to profile a Tensorflow-based solution while writing this article. We couldn't use a dedicated Python solution for this due to the C++ code, we were reliant on a solution created specifically for Tensorflow which unfortunately had a bug preventing us from interpreting the profiling results. In the case of Julia, profiling did not involve similar problems.

As our results show, it is possible to create a fast library for automatic differentiation while retaining the advantages of a high-level language. Although the running time of a CNN created on the basis of this library remains about 10% slower than one created on the basis of Tensorflow, basing the project on a single programming language has undeniable advantages.

VI. Ideas for further studies

Further research related to this library could include:

- Creating a CNN based on it designed for a different dataset and compare the results with the reference solution
- Performing profiling of the solution in Julia and the reference one in Python so as to see which operations exactly cause the time difference, where are bottlenecks and where our library performs better
- Closer look at the optimization of the library in Julia, while many recommended in Julia documentation methods where used there are still multiple areas to improve (like utilizing the arrangement of matrix data in memory, refactoring functions to remove unnecessary operations or getting rid of unnecessary types)

References

[1] O'Shea, Keiron, and Ryan Nash. "An introduction to convolutional neural networks." arXiv preprint arXiv:1511.08458 (2015).

[2] Innes, Michael. "Don't unroll adjoint: Differentiating ssa-form programs." arXiv preprint arXiv:1810.07951 (2018).

[3] Srajer, Filip, Zuzana Kukelova, and Andrew Fitzgibbon. "A benchmark of selected algorithmic differentiation tools on some problems in computer vision and machine learning." Optimization Methods and Software 33.4-6 (2018): 889-906.

[4] Margossian, Charles C. "A review of automatic differentiation and its efficient implementation." Wiley interdisciplinary reviews: data mining and knowledge discovery 9.4 (2019): e1305.

[5] Yu, Liang, Binbin Li, and Bin Jiao. "Research and implementation of CNN based on TensorFlow." IOP Conference Series: Materials Science and Engineering. Vol. 490. No. 4. IOP Publishing, 2019.

[6] Siddique, Fathma, Shadman Sakib, and Md Abu Bakr Siddique. "Recognition of handwritten digit using convolutional neural network in python with tensorflow and comparison of performance for various hidden layers." 2019 5th international conference on advances in electrical engineering (ICAEE). IEEE, 2019.

[7] Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." Advances in neural information processing systems 32 (2019).

[8] Gao, Kaifeng, et al. "Julia language in machine learning: Algorithms, applications, and open issues." Computer Science Review 37 (2020): 100254.

[9] Innes, Mike. "Flux: Elegant machine learning with Julia." Journal of Open Source Software 3.25 (2018): 602.