Using object-oriented languages in machine learning with artificial neural networks: a performance analysis

ABSTRACT

In the recent times, we have seen a large growth in usage numbers of various machine learning algorithms. As more research is done, more and more problems become solvable with some machine learning techniques. The practical question is how can one use these algorithms to extend and improve existing software. Artificial neural networks, one of concepts used for machine learning, have a strong background in mathematics. Because of this, environments like R and Matlab are definitely suitable for defining, training and using artificial networks. The reach of machine learning and artificial neural networks is far wider than the scope in which the mentioned environments are used however. In this paper we discuss the advantages and drawbacks of constructing and using artificial neural networks in modern object-oriented languages. Moreover, we investigate whether there is a performance loss or other disadvantages when using artificial neural networks in languages more suitable for all-round development.

INTRODUCTION

Artificial neural networks [1], often called just neural networks in computer science are a concept used in machine learning. They are modeled to resemble how biologists think the human brain works. There are nodes (neurons) which are distributed in layers and links between them (synapses). Every artificial neural network has to have at least the input and the output layer with some number of neurons in them. The usual case in modeling neural networks though is that there is also a hidden layer or a number of hidden layers. Artificial neural networks with many hidden layers are called deep neural networks (Deep Learning [2]).

As other machine learning concepts, teaching a neural network to compute something involves gaining a data set of inputs and expected outputs and feeding it to the neural network. When the network is initialized the connection weights are usually randomly set. Then an algorithm like Backpropagation [3] is used with the real data we have to modify those weights to get the desired results. We define an activation function for each layer (usually a sigmoid function) which is used to tell us whether the signals which arrive to a specific neuron are strong enough to activate it (pass the activation threshold). There are also several parameters we can modify like learning momentum or neuron bias.

There are many types of neural networks. The type mostly used is MLP (multilayer perceptron). In the more recent research several new types are being examined. Recurrent neural networks are neural network where connections between neurons can form a cycle. This can simulate internal states in the network or memory. Recurrent neural networks are used for handwriting recognition [4] and speech recognition [5]. Convolutional neural networks [6] are also a more recent model. Convolutional neural networks are mostly used in computer vision and natural language processing. They are modeled upon a visual cortex of animals. The difference is that a single input to a neuron also affects the neighboring neurons similar to how light stimulates animal's visual cortex. Another newer development are Neural Turing machines, a combination of neural networks and an external memory, which can deduct algorithms from input and output examples. [7]

Neural networks have become the obvious and sometimes the only choice to solving a number of real-world problems. Processing images, sound and other large data sets are all good candidates for artificial neural network application. Because of this there is a number of high quality frameworks for modeling, training, testing and using neural networks for a number of programming languages. Aside from being able to model neural networks in environments like MATLAB and R, programmers are now able to use neural networks in almost any software. The programming language most widely used is Python with frameworks keras, scikit-learn and PyTorch. There are also frameworks for Java, C#, golang, C++ and many other languages. When using any neural network toolkit or framework, the developer does not necessarily have to know how neural networks work, but can use them as a “black box” and feed it the learning dataset while trying different parameters.

In this paper we want to examine whether there is a performance penalty when using neural networks in general purpose languages opposed to those with a mathematical background like R and MATLAB. We will compare the same neural network models with same parameters in the following environments: MATLAB, R, Keras (Python) and gobrain (Go). We hope to prove that the performance loss is minimal and that these languages are suitable for machine learning concepts and that there is no real performance benefit to using MATLAB and R for neural network development.

RELATED WORK

As far as the authors of this paper now, there is no other research which provides insight into performance figures of general purpose languages for machine learning purposes compared to statistical or mathematical environments.

There is one artificial neural network implementation: FANN (Fast Artificial Neural Network)[1] which focuses on performance. There are also libraries for many languages which provide bindings for FANN which is written in C.

1 http://leenissen.dk/fann/wp/

TESTING DESCRIPTION

We chose the following environments to do our tests:

- R (with “nnet” package)
- MATLAB R2017a
- Keras library for Python
- Gobrain library for Go

The main comparison is between general purpose languages (Go and Python) and statistical environments (R and MATLAB). R and MATLAB were chosen as standard and most widely used tools for statistical and mathematical work. Keras for Python was chosen as one of the most popular neural network libraries for Python programming language. Python is widely used for machine learning purposes because of its scriptability, ease of use and variety of libraries available. We decided to also test gobrain library for Go language to see if there is a difference between programming languages which rely on built in virtual machines and those which do not.

It’s important to precisely say that we measure performance with the following parameters:

- Time to load training set
- Time to construct neural network architecture
- Time to train neural network with different number of epochs and parameters
- Time to compute a value with a trained network
- CPU time
- Peak RAM usage
- Total CPU cycles

For testing purposes we used a virtualized host (KVM on Fedora 25 Workstation) with these specifications:

- Intel® Core™ i5-4590 CPU - 1/4 cores allocated to the VM, using host CPU topology
- 8GB RAM allocated to the VM
- No GPU passthrough, running all tests on single CPU core
- Windows Server 2016

For profiling purposes SysInternals Process Explorer v16.21 was used.[2] Network was disabled on the guest operating system to stop various processes interfering (Windows Update for example).

Three very common test cases were used in all the environments. A simple XOR neural network and image recognition problem with MNIST dataset were chosen for classification tests. We wanted also to test not only classification but also regression

2 https://technet.microsoft.com/en-us/sysinternals/processexplorer

problems, so we used Boston house price dataset for that purpose.[3] The reasoning is we wanted to have one CPU intensive (10000 epochs), low memory problem which is a network for the XOR problem, one high memory, not CPU intensive problem which is a network for the MNIST dataset (running only one epoch is not CPU intensive) and one on the middle-ground – network for Boston house prices dataset. For MATLAB and R, we used the scripting capabilities of the environments to run the tests (running Rscript for R, and MATLAB with the appropriate command line arguments).

Testing neural network performance is difficult because the initiation process includes setting random values as initial weights inside the neural network. To mitigate this, every test has been run for a number of times (ten times) and then the averages were calculated. Every network and framework used the same network configuration and activation function. XOR network was trained through 10000 epochs, Boston through 50 epochs and MNIST through only 1 epoch of training.

The example code and complete results are located on the following address:

- http://github.com/nmilosev/annperformance

FEATURE PARITY

In the following table the features and characteristics of libraries used in this paper are given.

	Version	Underlying programming language	Multithreaded	GPU support
Matlab	R2017a	/	Yes	Yes
R	3.4.0	R	No	No (can be added)
Keras	2.0.4	Python	Yes	Yes
gobrain	1.0.0 git#735a9ae	Go	No	No

Table 1 – Used environments feature parity table

TEST RESULTS

Here we present the results of performed testing for all four tools solving all three problems. The results shown here are average over ten runes with each tool for solving each of the problems. Full results are available in the code repository.

3 https://archive.ics.uci.edu/ml/datasets/Housing

	TLOAD	TCONSTR	TTRAIN	TEVAL	PWS	CPUT	CYC
				KERAS			
XOR	0.9133015	0.1693748	52.9548774	0.0429978	41646.7	52.3699	166
Boston	0.6043903	0.1688212	71.8385816	0.2166496	41586.6	71.2617	225.9
MNIST	10.2801485	2.2045269	98.5193989	11.1632182	488250	118.9241	379
				GOBRAIN			
XOR	0	0	0.01464165	0	7585.6	0.02	0.0614
Boston	0.00218777	0	0.1966664	0.00224367	10237.2	0.1965	0.6043
MNIST	1.3115133	0.003504701	10.4041637	4.5133322	220134	15.9151	50
				MATLAB			
XOR	0.0026563	2.0347065	12.872168	0.1399652	379287.2	20.9465	66.6
Boston	0.0028105	2.2874952	1.5373677	0.1561935	372494.4	10.0247	31.6
MNIST	0.2507189	1.9281359	2.0602051	0.5404302	835342.4	10.2934	32.4
				R			
XOR	0.015	0.008	0.0101	0	47200.4	0.2908	0.8906
Boston	0.017	0.007	0.714	0.002	49254.4	0.984	3.0039
MNIST	7.006	0.004	23.836	0.776	3118109.6	31.7006	101.6

Table 2 – All test results (averages)

The columns in the table are as follows:

1. TLOAD – Time to load dataset, in seconds
2. TCONSTR – Time to construct the neural network, in seconds
3. TTRAIN – Time to train the neural network, in seconds
4. TEVAL – Time to perform an evaluation with trained neural network, in seconds
5. PWS – Peak Working Set, peak RAM usage in Kilobytes
6. CPUT – Total CPU time, in seconds
7. CYC – Total processor cycles, in Bilions of Cycles

First we will compare RAM usage, or PWS:

	keras (Python)	gobrain (Go)	MATLAB	R (nnet)
XOR	41646.7	7585.6	379287.2	47200.4
Boston	41586.6	10237.2	372494.4	49254.4
MNIST	488250	220134	835342.4	3118109.6

Table 3 – Peak Working Set comparison, peak memory usage in Kilobytes

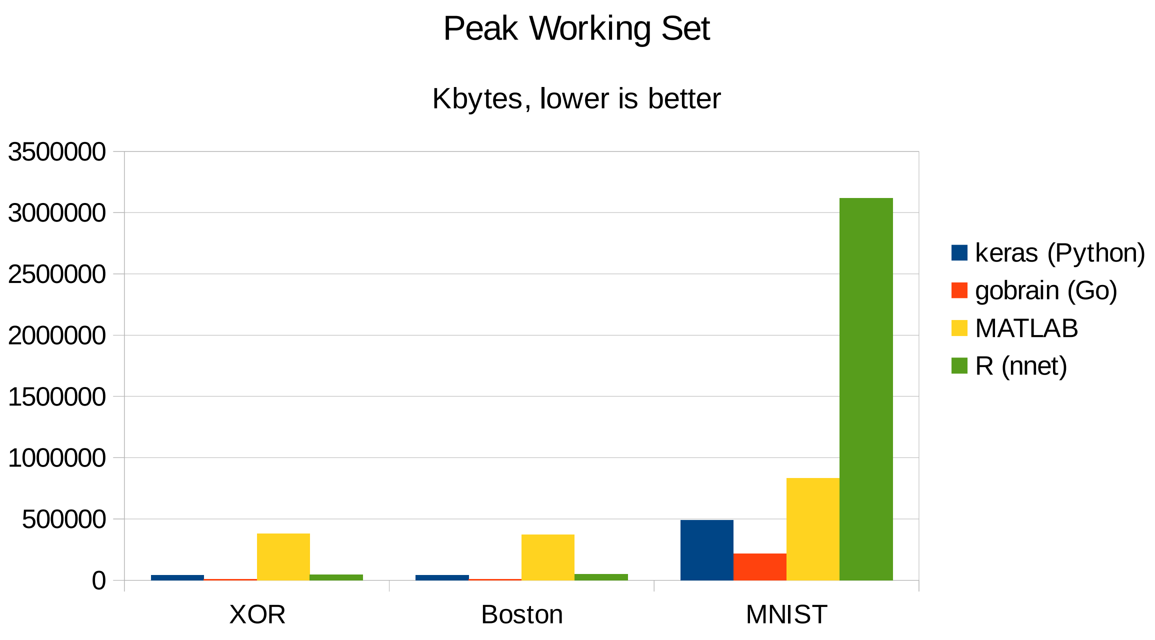

Figure 1 – Peak Working Set comparison

We observe that overall Go library gobrain uses the least amount of memory, while MATLAB and R use a larger amount of memory. This is probably the consequence of the overhead which MATLAB and R environments bring. We also see that while training the MNIST data set the largest amount of RAM was used, which was expected because of the scale of the neural network. We also observe a large amount of RAM used when training the MNIST dataset with R. In the testing R's data structure “dataframe” was used, which is less performant than other implementation which work better with sparse matrices. If running in environments where minimal RAM usage is needed, it is best to use gobrain.

Next, we compare total CPU time needed and number of cycle needed to perform data set loading, neural network construction, training and evaluation:

	keras (Python)	gobrain (Go)	MATLAB	R (nnet)
XOR	166	0.0614	66.6	0.8906
Boston	225.9	0.6043	31.6	3.0039
MNIST	379	50	32.4	101.6

Table 4 – CPU Time comparison (total seconds)

	keras (Python)	gobrain (Go)	MATLAB	R (nnet)
XOR	52.3699	0.02	20.9465	0.2908
Boston	71.2617	0.1965	10.0247	0.984
MNIST	118.9241	15.9151	10.2934	31.7006

Table 5 – Total CPU Cycles comparison (billions of cycles)

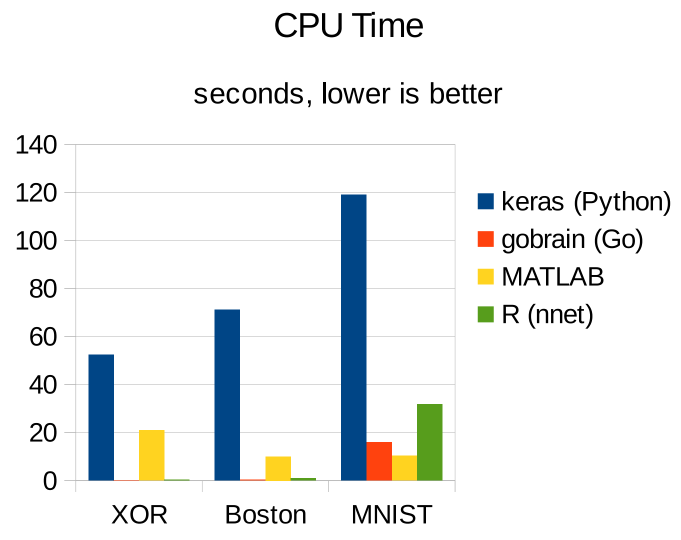

Figure 2 – CPU time comparison

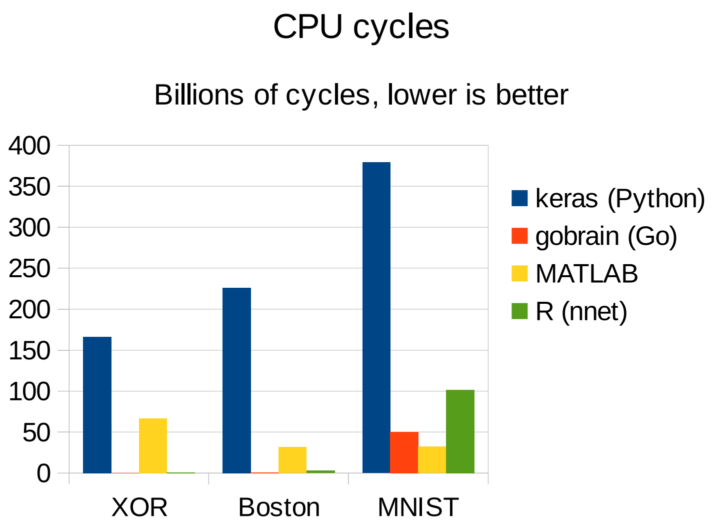

Figure 3 – CPU cycles comparison

We observe that keras library and MATLAB are using significantly more CPU time that the other two environments. The keras library was run without special "m2w64-toolchain" package, which uses a C++ backend which is the possible reason for its poor performance. The package was disabled to test true performance of the library without external help.

Lastly we compare maybe the most important statistic which is the reported training time:

	keras (Python)	gobrain (Go)	MATLAB	R (nnet)
XOR	52.9548774	0.01464165	12.872168	0.0101
Boston	71.8385816	0.1966664	1.5373677	0.714
MNIST	98.5193989	10.4041637	2.0602051	23.836

Table 6 – Total training time (in seconds)

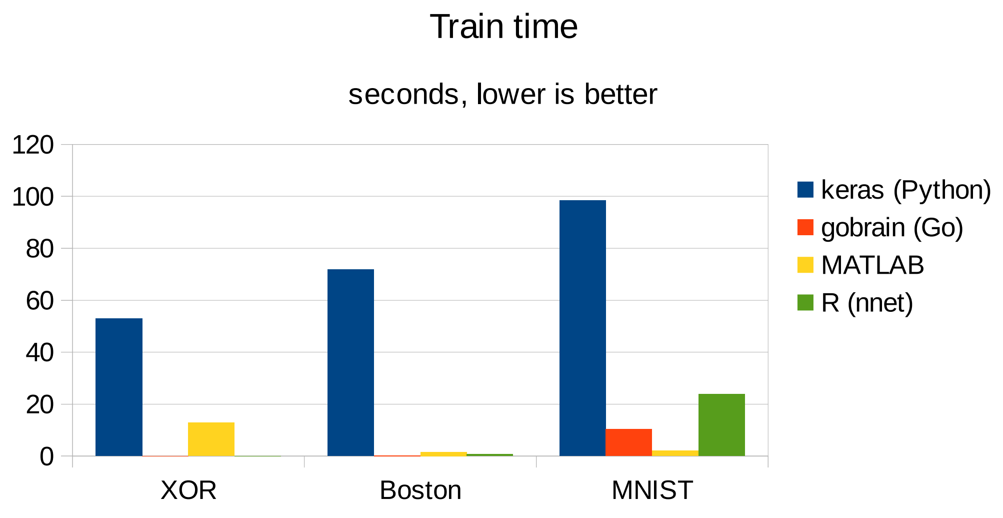

Figure 4 – Total train time comparison

We observe that gobrain, MATLAB and R perform similarly with only keras running much slower because of the disabled backend optimizations.

FUTURE WORK

We would like to also examine whether there is a significant difference with GPU supported training, although from an engineering standpoint the results should be comparable to our single-threaded results. Another interesting topic is to see whether different types of artificial neural networks (convolutional, recurrent, etc.) have environments where they perform better.

The end goal would be to investigate which environment performs best when the system resources are weak (for example in mobile and IoT devices) and develop a neural network framework for such devices. This way the data they already collect can be used for training immediately without sending it to more powerful servers.

CONCLUSION

In this paper, we investigated whether there are any downsides to using modern general purpose languages to perform many mathematical operations involved in training and using neural networks opposed to more mathematical and statistical software. The reason for this type of research is the fast and broad expansion of usage of machine learning solutions in different problem domains where the adoption rate is hindered with unfamiliar development environments. Our test results show that there is no (performance) downside to using familiar general purpose languages for machine learning concepts.

REFERENCES

1. Hopfield, John J. "Artificial neural networks." IEEE Circuits and Devices Magazine 4.5 (1988): 3-10.
2. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." Nature521.7553 (2015): 436-444.
3. Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." Neural Networks 1.Supplement-1 (1988): 445-448.
4. A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, J. Schmidhuber. A Novel Connectionist System for Improved Unconstrained Handwriting Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, 2009.
5. H. Sak and A. W. Senior and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. Proc. Interspeech, pp338-342, Singapore, Sept. 201
6. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
7. Graves, Alex, Greg Wayne, and Ivo Danihelka. "Neural turing machines." arXiv preprint arXiv:1410.5401 (2014).