The Journal of Open Source Education

mLEARn: An Implementation of Multi-layer Perceptron in C++

Kalu U. Ogbureke[1]

1 GRLMC, Rovira i Virgili University, Tarragona Spain

DOI: 10.21105/jose.00059

Software

- Review
- Repository
- Archive

Submitted: 22 May 2019
Published: 14 July 2021

Abstract

This paper presents `mLEARn`, an open-source implementation of multi-layer perceptron in C++. The techniques and algorithms implemented represent existing approaches in machine learning. `mLEARn` is written using simple C++ constructs. The aim of `mLEARn` is to provide a simple and extendable machine learning platform for students in courses involving C++ and machine learning. The source code and documentation can be downloaded from https://github.com/kalu-o/mLEARn.

Introduction

Artificial neural networks (ANNs) comprise a well-studied area of deep learning. This is because of their importance in many areas, from autonomous driving through speech technologies (Graves et al., 2013). The two main categories are usually recurrent (feedback) and feed-forward architectures (Bishop, 1995). Deep learning is currently an active research area in machine learning.

Statement of Need

Currently, popular deep learning frameworks are available, such as MXNet (Chen et al., 2015), Caffe (Jia et al., 2014) and TensorFlow (Abadi et al., 2016). Students often use these as off-the-shelf machine learning tools and have little or no control over the implementation. One of the reasons for this is because the codes are advanced and production ready. `mLEARn` addresses this and it can be used as an off-the-shelf machine learning tool. Furthermore, the coding style makes it easier to apply what was learnt in machine learning/C++ courses and extend the functionalities. This makes it easier to understand machine learning algorithms from first principles and extend the state-of-the-art.

Architecture of mLEARn

The classes implemented in `mLEARn` are Node, NetNode, Activation, CostFunction, Layer, Network, DataReader and Optimizer. The Node class is the fundamental data structure used; and NetNode is an extension of the Node class used for multi-layer perceptron.

The Activation class handles activation functions in the network. Currently, the functions implemented are sigmoid, tanh, rectified linear unit (ReLU), leaky ReLU, identity, softmax and exponential linear unit (ELU). The CostFunction class is responsible for objective/loss functions. Cost functions implemented are mean squared error (MSE), mean absolute error (MAE) and cross entropy.

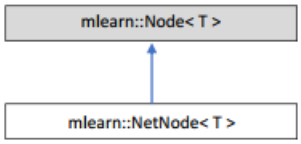

Figure 1: The Node class

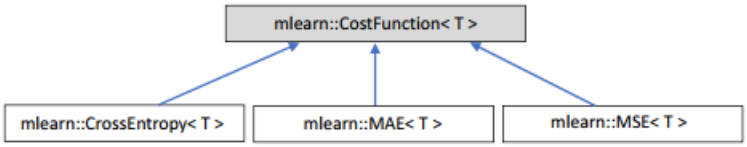

Figure 2: The CostFunction class

The Network class is a classic MLP consisting of sequences of layers, i.e., one or more hidden layers and an output layer. The DataReader class is the base class responsible for reading train/test datasets into features and labels. Three different readers are implemented, namely MNISTReader, GenericReader and IrisReader. The Optimizer class is the base class responsible for the training algorithms. Three optimizers are currently implemented: mini-batch stochastic gradient descent (Kiefer & Wolfowitz, 1952; Ruder, 2016), adaptive gradient (Adagrad) (Duchi et al., 2011) and root mean square propagation (RMSProp) (Tieleman & Hinton, 2012).

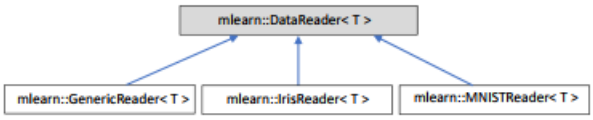

Figure 3: The DataReader class

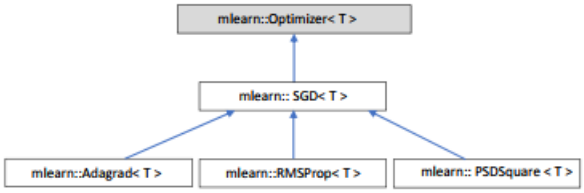

Figure 4: The Optimizer class

A number of new enhancements such as automatic differentiation, distributed computing and GPU support are planned to be added in the future.

Use in Teaching and Learning Context

The following describes some use cases in the context of teaching and learning, as example assigned student tasks.

1. Read a custom data set

This will involve extending the DataReader class in data_reader.cpp to read a custom data set. The learning objective is to enable students to extend the functionalities of a base class.

2. Activation functions

At the end of this exercise, students are expected to implement common activation functions and their derivatives. A number of activation functions such as Sigmoid, Tanh, etc. are already provided. Possible course assignments include: * Removing the implementations of these functions and asking students, for example, to implement the Sigmoid function and its derivative. * Extending with a novel activation function.

3. Understanding basic learning algorithms (back-propagation)

The back-propagation algorithm is a basic machine learning algorithm often taught in foundation courses in deep learning. Students should be able to Implement the forward and backward pass. These are already implemented in the Layer class.

4. Advanced deep Learning algorithms (RMS-Prop, Adagrad, etc.)

Adaptive learning algorithms are an active research area in deep learning. The Optimizer class provided contains an implementation of RMS-Prop and Adagrad. These implementations could be removed and students asked to implement these learning algorithms. Furthermore, others such as Adams, Adadelta, etc., are not implemented yet. Their implementation is a good exercise that will demonstrate grounding in adaptive learning algorithms.

5. Extending the architecture of the network (advanced)

The current implementation of the architecture needs improvement. One such area is refactoring the 'Layer' class into a base class and deriving sigmoid, tanh, relu and convolution layers from this.

Acknowledgements

The core work in neural networks and convergence was done as part of the DEA thesis "Training Multi-layer Perceptron using Genetic Algorithms," while the author was with GRLMC at Rovira i Virgili University, Tarragona. The author would like to acknowledge support for the speech and language technologies program.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., & Zhang, Z. (2015). MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, *abs/1512.01274*. http://dblp.uni-trier.de/db/journals/corr/corr1512.html#ChenLLLWWXXZZ15

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, *12*, 2121–2159. http://dl.acm.org/citation.cfm?id=1953048.2021068

Graves, A., Mohamed, A. R., & Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. *CoRR*, *abs/1303.5778*. http://dblp.uni-trier.de/db/journals/corr/corr1303.html#abs-1303-5778

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., & Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *Proceedings of the 22Nd ACM International Conference on Multimedia*, 675–678. https://doi.org/10.1145/2647868.2654889

Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, *23*, 462–466.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, *abs/1609.04747*. http://arxiv.org/abs/1609.04747

Tieleman, T., & Hinton, G. (2012). *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning.