ECBE Department
SIUC
April 3, 2022

ECE 572 - Neural Networks
Project

Andreas Karatzas - 856523415

Contents

Feed Forward Neural Network in C++ 17 and OpenMP for performance optimization

1.1 Abstract

In the project for course ECE 572, I will implement a feed forward neural network with sigmoid activation function.

1.2 Introduction

Artificial Neural Networks (ANNs) are used in wide range of applications including system modeling and identification, signal processing, image processing, control systems and time series forecasting. The baseline of those models is a special class called Feed Forward Neural Network[1, 2]. In this subclass of models, data is propagated forward only, and the model parameters are grouped in layers with no intra-communication. There is theoretically an infinite number of possible architecture regarding this type of models, hence they can have a lot of layers. In fact, Feed Forward Neural Networks were the beginning of a new Artificial Intelligence (AI) sub-field called Deep Learning, where the models used are designed to capture complicated patters. For that purpose, the model architecture as well as the dataset used to train them is very large. Consequently, in such cases the model performance poses a substantial challenge. There are several attempts on efficient ANN implementation using techniques, such as parallelization, to exploit the computational capabilities of the system architecture running a model[3]. Using low-level programming languages, such as C++[4], and frameworks that enable advanced parallelization and efficient data handling techniques, such as OpenMP[5], the programmer can achieve better results in terms of performance compared to most state-of-the-art Deep Learning frameworks[6].

For the project of course ECE 572, I will try to implement a feed forward model with sigmoid activation function. The user will have to make little to no configurations before the execution of the driver. It will be fully re-configurable regarding the architecture of the model. For example, the user will be able to define the layers and number of neurons using command line arguments. Moreover, the software architecture will follow that of well known deep learning frameworks, such as PyTorch[7]. That way the user will be able to easily navigate around the project if there is some prior experience with such frameworks. Furthermore, the user will be kept well informed throughout the data loading, the training and the inference process with *progress bars*. Finally, the advantages of the proposed project will be demonstrated using a well known dataset, Fashion MNIST[8].

1.3 Challenges

The implementation of a neural network in theory is an easy process. However, when it comes to putting together those formulas using software, the engineer is challenged to compile an efficient code implementation. The challenge becomes even greater when the project is carried out in a low-level programming language, such as C++, where the engineer has to solve substantial numerical and challenges since the project is based on scientific computation. After solving those challenges, the engineer has to structure the code in order to implement parallel software architectures and data pipelining for efficient execution. In this project, the optimization part will utilize the OpenMP framework for software parallelization and extreme device utilization.

1.4 Comparison

To actually realize the magnitude of optimization achieved by the parallel version of the project, I've implemented 2 more versions. The first is the *Python* version using *PyTorch*, which is the fastest deep learning framework in Python. The second is a `C++` implementation of the proposed feed forward neural network model running in serial mode. The *Python* version was implemented to capture the performance gap compared to the `C++` implementation. The results after training using the *Python* implementation can be viewed at figure 1.1. The results after training using the `C++` serial implementation can be viewed at figure 1.2. It is already clear that the `C++` implementation of the Feed Forward Neural Network is dominant. In fact the performance boost is above 1,000 %.

The code for those implementations can be found at `https://github.com/AndreasKaratzas/ece572`. The repository is currently private. If you would like to gain access please email at andreas.karatzas@siu.edu with your GitHub account, and I will add you as a collaborator.

```
Device utilized: cpu.

model(
  (input_l): Linear(in_features=784, out_features=150, bias=True)
  (hidden_1): Linear(in_features=150, out_features=100, bias=True)
  (hidden_2): Linear(in_features=100, out_features=50, bias=True)
  (output_l): Linear(in_features=50, out_features=10, bias=True)
)
[EPOCH    0] [LOSS 0.09133] [ACCURACY   5943 out of  60000] Work took   72.8 seconds
[EPOCH    1] [LOSS 0.09140] [ACCURACY   6549 out of  60000] Work took   72.5 seconds
[EPOCH    2] [LOSS 0.09221] [ACCURACY  13070 out of  60000] Work took   78.6 seconds
[EPOCH    3] [LOSS 0.07209] [ACCURACY  37415 out of  60000] Work took   80.9 seconds
[EPOCH    4] [LOSS 0.01148] [ACCURACY  51379 out of  60000] Work took   80.0 seconds
[EPOCH    5] [LOSS 0.00217] [ACCURACY  53666 out of  60000] Work took   80.8 seconds
[EPOCH    6] [LOSS 0.00079] [ACCURACY  54865 out of  60000] Work took   76.3 seconds
[EPOCH    7] [LOSS 0.00040] [ACCURACY  55687 out of  60000] Work took   72.9 seconds
[EPOCH    8] [LOSS 0.00020] [ACCURACY  56302 out of  60000] Work took   72.3 seconds
[EPOCH    9] [LOSS 0.00012] [ACCURACY  56811 out of  60000] Work took   73.5 seconds
C:\Users\andreas\anaconda3\envs\mnist-fcn\lib\site-packages\torch\nn\_reduction.py:42: UserWarning: size_average and red
uce args will be deprecated, please use reduction='sum' instead.
  warnings.warn(warning.format(ret))
[EVALUATION] [LOSS 0.08758] [ACCURACY   9430 out of  60000] Work took    4.9 seconds
```

Figure 1.1: Results from the Python implementation.

```
Loading training dataset       |**************************************************|  100%

Loading evaluation dataset     |**************************************************|  100%

Neural Network Summary:        [f := Sigmoid]
------------------------------------------------------------
Layer [1]        785 neurons
Layer [2]        151 neurons
Layer [3]        101 neurons
Layer [4]         51 neurons
Layer [5]         10 neurons

[EPOCH    1] [LOSS 0.15253] [ACCURACY  47246 out of 60000] Work took    7 seconds
[EPOCH    2] [LOSS 0.11542] [ACCURACY  50498 out of 60000] Work took    7 seconds
[EPOCH    3] [LOSS 0.10392] [ACCURACY  51505 out of 60000] Work took    7 seconds
[EPOCH    4] [LOSS 0.09715] [ACCURACY  52041 out of 60000] Work took    7 seconds
[EPOCH    5] [LOSS 0.09407] [ACCURACY  52363 out of 60000] Work took    7 seconds
[EPOCH    6] [LOSS 0.09024] [ACCURACY  52704 out of 60000] Work took    7 seconds
[EPOCH    7] [LOSS 0.08922] [ACCURACY  52668 out of 60000] Work took    7 seconds
[EPOCH    8] [LOSS 0.08407] [ACCURACY  53162 out of 60000] Work took    7 seconds
[EPOCH    9] [LOSS 0.08109] [ACCURACY  53381 out of 60000] Work took    7 seconds
[EPOCH   10] [LOSS 0.08000] [ACCURACY  53488 out of 60000] Work took    7 seconds

[EVALUATION] [LOSS 0.09077] [ACCURACY   8767 out of 10000] Work took    0 seconds

Benchmark results: 106.89459 seconds
```

Figure 1.2: Results from the serial C++ implementation.

Bibliography

[1] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[2] M. Sazli, "A brief review of feed-forward neural networks," *Communications, Faculty Of Science, University of Ankara*, vol. 50, pp. 11–17, 01 2006.

[3] A. A. Huqqani, E. Schikuta, S. Ye, and P. Chen, "Multicore and gpu parallelization of neural networks for face recognition," *Procedia Computer Science*, vol. 18, pp. 349–358, 2013. 2013 International Conference on Computational Science.

[4] B. Stroustrup, *The C++ Programming Language*. Addison-Wesley Professional, 4th ed., 2013.

[5] L. Dagum and R. Menon, "Openmp: An industry-standard api for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, pp. 46–55, jan 1998.

[6] H. Jang, A. Park, and K. Jung, "Neural network implementation using cuda and openmp," *2008 Digital Image Computing: Techniques and Applications*, pp. 155–161, 2008.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[8] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017.