A Beginners Guide to Perfomance Factors of Machine Learning Learning Applications with Focus on Neural Nets and Deep Learning

Johannes Wünsche
Otto-von-Guericke University Magdeburg
Email: johannes.wuensche@st.ovgu.de

Abstract—**Machine Learning is probably the most present topic of computer science in the media. This popularity only increased over the recent years thorugh widely publicly known apllications ranging from DeepBlue(not so recent) to IBM Watson and AlphaGo. Because of these successes Machine Learning is sometimes viewed as omnipotent solution to every imaginable problem. This is most certainly not the case, Neural Nets and alike are powerful but require a large amount of maintenance**

I. Introduction

Machine Learning seems to be everywhere around us. With the recent success of Deep, Convolutional,and Neural Networks, and other Machine Learning tools, especially the, in general media, well reported Learners like AlphaGo, Silver or Watson, these are experiencing a kind of renaissance their usage has spread from industrial(e.g. Self-driving cars rely on autonomous interpretation of camera feed and sensor output that can be optimized by using machine learning) to retail. But also in modern automation and research, machine learning becomes a greater field managing to find more efficient ways to achieve better results in many expertise.

In this paper we want to observe and identify the most performance influencing factors of theses networks while designing and implementing them for specific applications for both small and large scales. We expect that this study will provide help for readers interested in learning about the requirements for managing data and operations of an efficient neural network.

TODO: related surveys

At first we attend the choice of neural network and the impact of this decision. Afterwards we chose to have a look, on one hand, at small scale neural networks running on a single machine without large data sets and a low query frequency after the initial training, and at the other hand, large scale neural networks operating with more layers and nodes, large data sets and distributed system to handle the computational power required by the neural network training algorithms to be completed in reasonable time.

II. Background

The reader is advised to be familiar with the base idea of a neural network and some common types of networks like the deep, convolutional, feed forward and recurrent networks.

The most common neural network frameworks which we, because of their wide popularity, will mostly view are Tensorflow [**?**], Caffe [**?**], Torch [**?**], CNTK, deeplearning4j, Keras, MXNET. These frameworks have been benchmarked in numerous settings and applications and can help us show performance influencing factors.

III. Choice of Neural Nets

what net when to use, and starting values of neural nets like amount of layers and starting weights

IV. Choice of Processing Unit

The next important choice is the choice of the actual operating calculation unit. We distinguish them into three main models, CPU, GPU and GPU-Clustering.

A. CPU

As the first most naive option we looked on the CPU for training of the neural net with the simple single thread and more complex multi-thread calculation.

1) Single-thread: The most naive implementation is the single-thread CPU calculation, because oft this more brute force approach and sequential execution and calculation this is also the slowest and with a larger number of connected nodes simply unfeasible because of the excessive calculation time required . But as can be thought of this does not uitilize the full capacity of the CPU and therefore results in performance loss in comparison to multithread usage(Fig. 1).

2) Multi-thread: A more complex approach consist of utilizing the multi-core characterisitics of modern CPUs, by splitting of training calculation with only requirements to already calculated values. This is for example preferrable when training with the help of a *Backpropagation* algorithm. The effect continues to grow by using more threads but reaches it limits when using more threads than physical cores available.

Though this approach is limited by the actual core number available and should not overstep the number of physical cores, like with intel *hyper-threading* technology, which can lengthen the required calculation time because of a more inefficient usage [**?**].

Framework	1 Thread	2 Threads	4 Threads	8 Threads
Caffe	1.324	0.790	0.578	15.444
Tensorflow	7.062	4.789	2.648	1.938
Torch	1.329	0.710	0.423	na

Fig. 1. Training time(in s) for a mini-batch of size 64 of a Fully Connected Network(FCN) trained with synthetic data on a i7-3820 with 4-physical cores using 3 different commonly used Deep Learning frameworks [?]

B. GPU

A more advanced implementation is to use the shared memory, multi-core environment of modern GPUs. For example with NVIDIAs CUDA this can be done efficiently and result in a speed-up of calculation of up to 3 times the CPU-based approachs performance for deep learning algorithms over optimized floting-point baseline [?]. Allmost all recent neural net learning frameworks support this calculation unit because of its excellent performance rating for neural networks.

Problems lie in the tightly restricted memory of graphic cards that may not be able to contain all nodes with their corresponding weights as well as training dataset. This leads to some communication overhead depending on the communication strategy chosen.

C. GPU-Cluster

The next logical step is to use multiple GPUs further reducing training time but also creating new challenges, because of the decentralized nature of the Cluster, like communication overhead. This is mostly used in large scale applications and is most of the time not needed in small scale.

A main advantage of the Cluster is of course the increased computation power available during the training phase, reducing training time by 35% when doubling the number of GPUs in CNTK and MXNET, 28 % in Caffe and 10 % in Torch and Tensorflow [?].

Another advantage is the greater availability of memory somewhat easing the restrictions of the memory restricted GPUs structure. While this is an advantage it is not efficient if used as the single goal of multiple GPU usage because of the high price connected to acquiring them.

Fig. 2 visualizes this improvement showing the reduction of training time steadily with the increasing of amount of GPUs used. Although costly this leads to a constant improvement. Higher Numbers of GPUs can be again slower or offer no significant improvement if used the same as smaller amounts because of an increasing communication overhead and bottleneck in the CPU, which is responsible for managing seperate GPUs.

1) Multiple GPU-Clusters: Taking a step further it is possible to connect multiple Clusters of GPUs to compute a single task. In doing this we again increase raw computation power, but also increase the total amount of communication required.

V. Scaling challenges

Here I want to clarify a bit on strategies to create an efficient CPU-GPUs interchange with as little overhead as possible. TODO: Spanish thesis on SuperComputer Learning

VI. Conclusion

yeah ... pretty much the conclusion shortly repeating the basis that is important for it and point the way they were created, should be larger part to emphasize the importance of some factors over others and the reasoning behind this classification [?]

TODO: Takeaway, different ideas, innnovative

Acknowledgment

The author would like to thank Gabriel Campero Durand for advise and help to write the paper

References

[1] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," *arXiv preprint arXiv:1608.07249*, 2016.