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 and Related Work

As a small scale neural network applications we define a network with few neural nodes inside the network and because of this rather limited size most of the time run on a single machine and or single processing unit(worker) which can complete the training set in a reasonable time.

On the other hand as a large scale neural network applications we define a network with a large number of nodes which can because of the complexity of the resulting connections take a large amount of time to complete training if computed on one worker and therefore will take use of a distributed system consisting of multiple workers and one or more servers to synchronize the workers and distribute the required data.

Most commonly used Neural Networks of the more classic single or non hidden layer kind are Feed Forward, Auto Encoders and Restricted Boltzmann Machines, which all have only a simple layer of hidden nodes and therefore are quite restricted in what they can compute based on the input data.

Feed Forward Networks(FFN) simply connects input, hidden and output layers with an all to all connection between the layers with the number of outputs being equal or smaller to the number of hidden nodes.

Auto Encoders(AE) are of an similar build but the hidden layer compresses the input information into fewer nodes while these are later again decompressed to output nodes that match the previously given input nodes.

Restricted Boltzmann Machines(RBM) only have an input and an output layer with an all to all connection between these. This RBM has in comparison to the normal Boltzmann Machine a better training behaviour which leads to a faster trained network.

A more advanced architecture is to increase the number of hidden layers to enlarge the application possibility of these networks for more complex models these are then called Deep Networks. The most commonly used ones are Variational Auto Encoders , Deep Belief Networks, Generative Adversial Networks, Recurrent Neural Networks ,e.g. Long short-term memory networks, Deep Convolutional Neural Networks, Deconvolutional Networks, and Recursive Neural Networks.

Variational Auto Encoders(VAE) consists of a encoder, sampler and decoder and are able to to generate output based on specification given to the VAE.

Deep Belief Networks(DBN) are a combination of RBM and FFN in which the RBM are used to pretrain the network with the initial training data and the FFN to adjust the learned values slightly to improve the result.

Generative Adversial Networks(GAN) like the VAE creates random output based on the model found in the given training data. VAE are a specialization of GAN because of the restrictions that can be set for the output which is not possible in a GAN.

Deep Convolutional Networks(DCN) are based on Convolutional Nodes compressing the incoming information into the needed base model and then calculate the required output. [1] They are used to abstract input information into more general models.

Deconvolutional Networks(DN) [2] are less surprisingly the opposite of DCN in the idea that input information is first spread to a larger amount of nodes and then processed by the following Nodes to be used by the output.

Long short-term memory(LSTM) [3] are a variant of Recurrent Neural Networks including additionally memory cells and gates that control the content of the cells. This model has been able to improve learning process of RNN and results.

The reason why the training of neural networks is so expensive can be traced back to the algorithm used to train all commonly used networks, which is backpropagation. Backpropagation is based on the idea that we calculate the deviation of the output values from the actual result and then use this backwards thorugh the layers of the net to calculate a correction,based on the error of the node and the learning rate, for each connection weight until the input layer is reached [4]. Some Networks like DBN use a slightly diverged "Gentle" Backpropagation, which uses a lowered learning rate after the initial values have been found from the training data with the help of the pretrain RBM [1].

Another performance influencing factor besides the actual neural net and the used learning algorithm is the amount of data required for the net to be trained and the size of the input of each training example.While this is mostly important for large scale applications it can also be hindering for smaller ones. For example if we use a Convolutional Neural Network to classify the content of a greyscale picture we get for small $512 * 512$ sized images 262144 inputs that each need to be transferred to the processing unit, CPU or GPU, for each example.

This large amount of data can for one take some time to be transferred, if for example this data package is required by an external processing unit like a GPU other units can not receive new data and therefore lose performance due to them being idle. Also if too much data is needed in relation to the actual output the data storage can achieve enduring performance loss will occur.

Further can the required training data exceed direct accessible memory from the processing unit, enhancing the problem of delivering data to all processing units.

Additionally the synchronization can be somewhat difficult and extensive for large neural networks on multiple processing units because of the interchange of new weights calculated between them after a certain number of iterations on their training cases.

The most common neural network frameworks which we, because of their wide popularity, will mostly view are Tensorflow [5], Caffe [6], Torch [7], CNTK [8], deeplearning4j [9], Keras [10], MXNET [11]. These frameworks have been benchmarked in numerous settings and applications and can help us show performance influencing factors.

TODO

The topic of performance of neural networks and influencing factors has been touched by many benchmarking survey like [12] and [13]. But we hope to offer a collected view on the most influencing factors in a more abstract style.

III. Choice of Neural Network

The first important choice is to what neural network is going to be used. This decision is heaviliy reliant on the application that is to be implemented. This decision influences all further actions and has to be carefully made, to assist this decision we will offer some generalization when to use simple neural networks or deep neural network and what specializations of certain neural networks are giving them advantage while doing specific tasks.

A. What kind of networks to use

1) Simple Networks: Simple task can be done by a network which is not deep like a FFN or a RBM. They are faster to train and even larger training sets can be completed in a reasonable time without multithreading or usage of GPUs. But there small size can lead that accuracy decreses when the model becomes more difficult than anticipated. Small scale applications often incur that a simple neural network is used because of the smaller training set and shorter computation time.

2) Deep Networks: More complex tasks more often require a deep neural network that can learn a more complex model than simple networks and offer a greater accuracy while doing this. The major downside of these networks that they require a larger amount of time to train and can be oversized for the actual to be represented model, and therefore adding unnecessary computation time. Large scale applications mostly contain deep networks because of their larger training sets and more complex models underlying the actual task.

We can not offer a clear answer when to use which kind of network, every problem must be individually analyzed and based on the information, and complexity, as well as the acceptable error rate of the final network the decision has to be made. For cases that are unclear a simple network can first be constructed,because of their short training time, and if the result is unsatisfying a more complex network can be considered.

An also important choice is the setting of the initial values of connection weights, here it is advisable to start at low values like 0.1 to prevent straying too far from the optimal solution, since algorithm like backpropagation mostly find a local optimum initial values too far of from the global optimum will deliver worse results.

B. Simple network Nodes

If a simple network shall be used the next choice will be the amount of nodes in the hidden layer. This is of course dependent one foremost the number of inputs and outputs, the number of hidden nodes should deviate to strong from them. How strong this deviation really is is dependent on the chosen neural network, for example a feed forward network profits when the number is equal or larger than the number of inputs, while an auto encoder probably has fewer hidden nodes. For a simple network this decision can be if too far from the optimum less fatal than for a deep network, while a simple network may lose some accuracy or gain additional training time a deep network can experience these effects at a greater scale.

C. Deep networks Nodes

Like in simple networks the amount of nodes to be used is an important value that determines the final ability of the network to fulfill its task in the required accuracy and speed. The acutal number of nodes chosen at the end is of course a bit more complex in deep networks for example networks like DCN require a special node configuration based on their basic idea(step by step decreasing of nodes per layer until desired compression reached) , or DN(opposite of DCN). So how the node configuration looks like is strongly dependent on the net chosen.

D. Specific deep networks

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 [12].

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 [12]

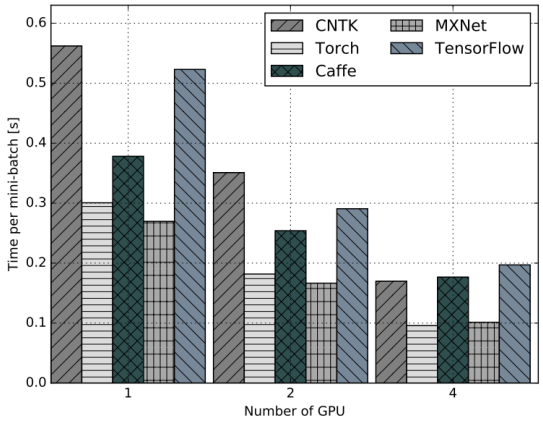

Fig. 2. Time per mini-batch in s compared between different amount of GPUs used [12]

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 [12]. 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 [12].

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

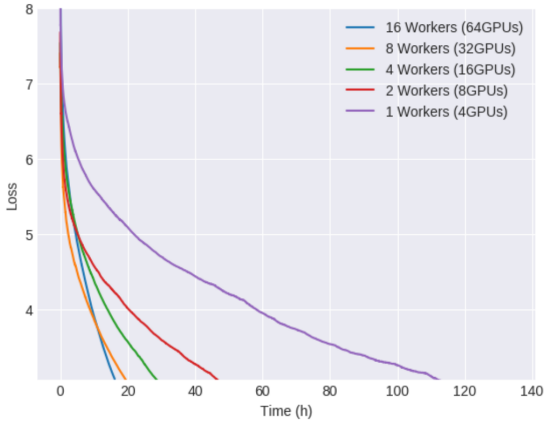

Fig. 3. Time in h of different asynchronous configurations until reaching destined error rate(loss) [14]

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. Similar to multiple GPUs this leads at the beginning with 2 or more Clusters to a stronger improvement than later on with 8 to 16 Clusters (Fig. 3)

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 [12]

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] J. Patterson and A. Gibson, *Deep Learning: A Practitioner's Approach*. Beijing: O'Reilly, 2017. [Online]. Available: https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/

[2] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *Neural Networks, 1993., IEEE International Conference on*. IEEE, 1993, pp. 586–591.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[6] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[7] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.

[8] Microsoft/cntk. Access Date: 06.01.18 11:55:34. [Online]. Available: https://github.com/Microsoft/CNTK

[9] S. team Chris Nicholson. Deeplearning4j.org. Access Date: 06.01.18 11:50:00. [Online]. Available: https://deeplearning4j.org/

[10] Keras.io. Access Date: 06.01.18 12:00:17. [Online]. Available: https://keras.io/

[11] Mxnet: A scalable deep learning framework. Access Date: 06.01.18 12:02:43. [Online]. Available: https://mxnet.apache.org/

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

[13] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," 2016.

[14] F. Sastre Cabot, "Scalability study of deep learning algorithms in high performance computer infrastructures," Master's thesis, Universitat Polit'ecnica de Catalunya, 2017.