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 general media. This popularity only increased over the recent years thorugh widely publicly known applications 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 design and implementation work. We want to give a short introduction to performance influencing factorts of neural networks during training and the resulting network.**

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 scale applications. We expect that this study will provide help for readers interested in learning about the the initial design of neural networks, 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. We divided our survey into first the choice of neural network afterwards choice of processing units, including basic data splitting strategies for multi processing units systems, followed by challenges occuring when using large and small scale applications.

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(RNN) 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.

The next major performance influencing factor will be the choice of the actual processing unit. This decision is the most computation time influencing and can also influence the quality of the result. We divide our processing unit into three groups starting with the CPU,which have a greater memory capability because they can be equipped with additional RAM sticks but posses a smaller virtualization limit than GPUs because of their differentiating architecture, then the GPU,with a much larger architectural focus on multithreading than CPUs, and following GPU Clusters,being simple a seperate unit consisting of multiple GPUs, like they are used in supercomputers.

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. Also it is advisable to use a deep network if the data is either of image or audio nature and/or needs to include changes over time of an input signal [1].

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

Based on the content of the task a few networks can be more fitting than others to fulfill it. To deliver a shot overview of it we use a classification by Patterson [1] for the basis of our advise.

If the task includes the generation of data to be output a GAN, VAE or RNN/LSTM. Which of these to choose is strongly dependent of the specific parameters of the task. For example if we want to generate random new data following the model of our training data the closest to this would be a GAN because it is by design designated to fulfill such tasks. But if new data which is not completely random is to be generated a VAE can be considered which offers by design this possibility. In practice there are lots of generation examples available using these networks for example image generation with GAN [14] (corresponding paper [15]) or text generation with LSTM [16] (paper describing base approach [17]).

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

TABLE I
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]

If the task includes classification or interpretation of visual data neural networks like CNN or DBN can be used. For once CNN are a straightforward choice since they are able to compress data and therefore abstract smaller data like the content of an image from the original image. But DBN offer also an possibility to complete the task with the major adavantage of being able to first roughly guess values from the training data and then train unsupervised in the fine-tuning phase as shown by Zhong 2016 [18].

If the task includes the classification or intepretation of incoming data over time the ability of RNN and LSTMs to data that is of sequential nature comes in handy. For example voice recognition is possible to implement with the help of LSTMs as shown by Soltau 2016 [19].

IV. CHOICE OF PROCESSING UNITS

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 have a look 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(Table I).

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].

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. Multi-GPU

To further improve performance during training multiple GPUs can be used to train the neural net. This shortens training time but also requires new actions like synchronizations of GPUs and splitting of data and/or neural network.

A main advantage of Multi-GPU 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 if used as the single goal of multiple GPU usage because of the high price connected to acquiring them.

Fig. 1 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) Parallelism between GPUs: There are two kind of parallelism approaches applicable to the problem, first there is data parallelism which splits the training data into multiple sets, mostly used when the training set is splitable into multiple subsets. This can be done by dividing the data by case or by feature. Second model parallelism in which the net is divided into separately runnable parts which later can be joined together again, this method is fitting for single system distribution in which the GPUs share one PCIe bus, but lead to the result being more vulnerable to calculation failures [20].

2) Synchronization between GPUs: For synchronizing Multi-GPU setups like in Figure 3 there are theoretically two choices. One being the synchronous method in which after a value is calculated they are shared with every other GPU. This method is applicable to all Multi-GPU configuration and delivers an acceptable result [21]. The update of values for all GPUs occurs only after the updated values have been send by each of them. Until a large amount of over 40 GPUs are

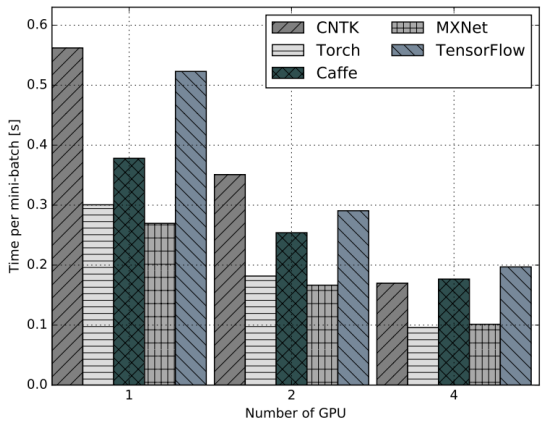

Fig. 1. Time per mini-batch in s compared between different amount of GPUs used [12]

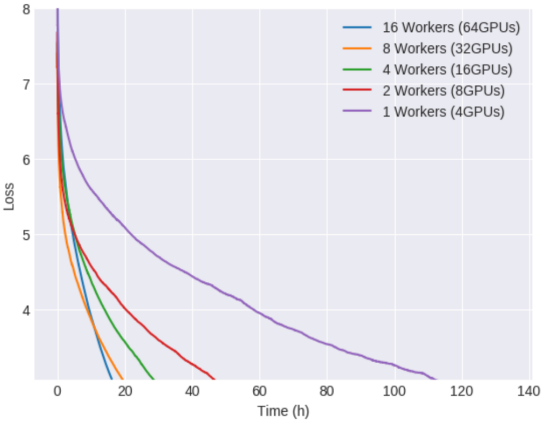

Fig. 2. Time (in h) of different asynchronous configurations until reaching destined error rate(loss) [20]

used this mehtod can be used without limitation [20]. There is also the possibility to run the GPUs asynchronously, by letting them use different versions of the values and only updating them periodically by sending an update request to the CPU. This can lead to a slight improvement of the training speed but is not guaranteed depending on the training model [21].

D. GPU-Clusters

The next logical step is to use multiple Multi-GPU systems 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.

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 as seen in (Fig. 2)

2) Parallelism of multiple GPU-Clusters: Like the Multi-GPU setup this can be done by creating either data or model

parallelism. Combination are also possible when using GPU-Clusters, for instance in a setup consisting of multiple Clusters and one single communication server a model parallelism can be used to distribute the model on the different clusters, but they internally use data parallelism to split work [21].

3) Synchronization of multiple GPU-Clusters: Similar to parallelism the synchronization of GPU-Clusters can be undertaken either with the synchronous or asynchronous like in Multi-GPU configurations. Again these tactics can be combined to create a mixed-asynchronous strategy, using an asynchronus training on inter-clusters level and cluster internal synchronous training [20]. This creates the best performance on large configurations with many clusters [21].

V. Large scale challenges and Small scale challenges

Dependent on the choice of scale each one has its own challenges to be considered during design and implementation of the neural network.

A. Small scale challenges

First and foremost it is important to notice that the biggest restrictions of small scale applications is the limited computation power available during the initial training process, later during usage of the net this is less impeding because either the net is completely pretrained and not changed later or the only change is some optimization when the network receives a query.

This leads to the focus lying on the design to find the best fit of complexity to fit the desired model. Otherwise longer training time or less accuracy of the resulting network will conclude.

Further small scale applications are relatively free of performance related complications, because of their relative simplicity and fast modern computer hardware enabling operations which are not optimal to be resolved in a non less desirable time.

B. Large scale challenges

Opposing to small scale applications large scale ones to not have the actual processing power as their highest pending restriction but the communication and data transfer between all acting processing units. This problem occurs because a larger data flow has to be managed to more processing units compared to small scale applications.

The data flow is a problem on multiple fronts. For one the transfer of the training data from main memory to GPU memory can be the first bottleneck restricting faster processing speeds from being realized. For example if the GPU only receives data if no calculation is currently active this slows down the training process unnecessarily. To increase the speed of the calculation data can be fetched during the processing of already present data, this is called data prefetching as described and implemented by Yang 2010 [22]. This solution can solve the problem partially but reaches it limits if used with multiple GPUs for example configuration (Fig. 3), which

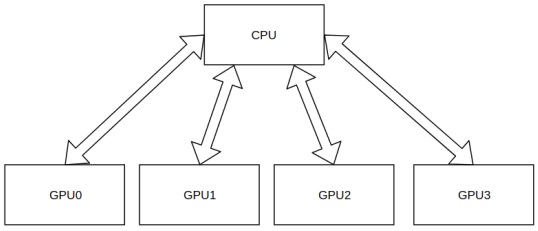

Fig. 3. One example configuration consisting of a CPU and 4 connected GPU units for training the neural network.

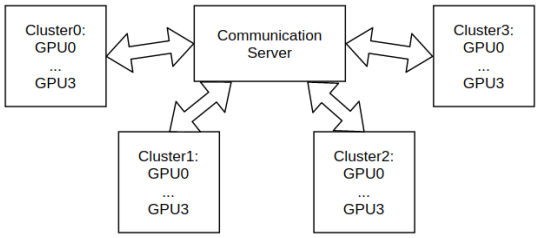

Fig. 4. One example configuration consisting of a communication server and 4 connected GPU-Clusters each containing 4 GPUs.

leads that the CPU will become a bottleneck if the training data of each individual GPU is too large to fit in the GPU memory, since the CPU can only manage transfer to the GPUs one at a time. For example a GPU equipped with PCI-Express 4.0 with a x16 Throughput can at best receive 31.5 GB/s, if we assume that all training data can be fit into main memory, that way if we need to refresh the complete memory of a modern GPU like NVIDIA GTX 1080, we need 0.253968 s, so to refresh all GPUs memory without including any overhead produced by switching operations between the GPUs we have an absolute transfer time of 1.015873 s, and for 3 units 0.7619 s, so if our GPU completes the desired calculation faster, for example 0.5 s, we have an 0.2619 s period in which our GPU idles every iteration, assumed that we can transfer all data during the calculation time. Therefore we have an effective calculation computation time each phase of 65.62 %. This problem only enlargens if the main memory is to small to fit the complete training data since connections to solid state disk are too slow and hard disk do physically do not reach the required throughput.

Additional data flow is created when synchronizing GPUs with each other, by approaches described beforehand in "Choice of Processing Units", they each have their own advantage but can limit efficiency in certain situations.

Synchronous training, while being recommend for setups that have 40 GPUs or less, can create idling time if one unit is slower than others, since all updates have to be send to the server before the next iteration can be undertaken this slows down all other units. This problem becomes more probable the more units are used. Also more GPUs lead to more communication if synchronization is required after very step, so they increase the amount of data managed by the CPU

or communication server. This applies to configuration like Figure 3 and Figure 4.

If an asynchronous training approach is used problems may occur influencing the result, if updates are to sporadic and different versions of values drift too far apart. To avoid this problem GPU-Cluster configurations do not rely on complete asynchronous training startegies, but on the already described mixed-asynchronous minimizing this problem while still reducing possible communication overhead.

To ease the effect of GPU to GPU communication,by relieveing the CPU of managing the , technologies like GPUDirect from NVIDIA [23] can be used allowing direct GPU to GPU communication if they are connected to the same PCIe bus, so this is only appplicable within a GPU Cluster or a single system.

VI. Conclusion

During our survey we observed that the most influencing factors are the basic choice of neural network to be used, this includes its complexity as well as the actual architecture the network is build of. Further the processing unit that is chosen has a strong influence on the quality and speed the final network that lies within acceptable error limits will be reached. Based on the choice of this other factors come into consideration influencing the process like communication between processing units and parallelism of data between them. They are not quite as influential as the first two but can lead to a strong improvement or worsening.

Especially data transfer time and alike carries a potential to lengthen the training time of an significant amount. To ease this technology like GPUDirect [23] and data prefetching [22] can be used. It may be possible to decrease the effect of the data flow further by introducing a memory transfer unit which can indepently organize data and can handle parallel transfers to and from multiple units. This could decrease the calculation time to a bare minimum but would requires additional hardware and maybe modification of the used GPU and CPU. Although this would break security of the system on broder range because of the unauthorized memory access and changes, it could offer a steep improvement for large calculations on distributed systems.

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] Junyanz, "junyanz/igan," Apr 2017, access Date: 07.01.18 10:58:57. [Online]. Available: https://github.com/junyanz/iGAN

[15] J.-Y. Zhu, P. Krähenbühl, E. Shechtman, and A. A. Efros, "Generative visual manipulation on the natural image manifold," in *European Conference on Computer Vision*. Springer, 2016, pp. 597–613.

[16] Spiglerg. (2017, Dec) spiglerg/rnn_text_generation_tensorflow. Access Date: 07.01.18 11:04:34. [Online]. Available: https://github.com/spiglerg/RNN_Text_Generation_Tensorflow

[17] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[18] P. Zhong and Z. Gonga, "a diversified deep belief network for hyperspectral image classification," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 443–449, 2016.

[19] H. Soltau, H. Liao, and H. Sak, "Neural speech recognizer: Acoustic-to-word lstm model for large vocabulary speech recognition," *arXiv preprint arXiv:1610.09975*, 2016.

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

[21] W. Wang, G. Chen, H. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K.-L. Tan, S. Wang, and M. Zhang, "Deep learning at scale and at ease," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 12, no. 4s, p. 69, 2016.

[22] Y. Yang, P. Xiang, J. Kong, and H. Zhou, "A gpgpu compiler for memory optimization and parallelism management," in *ACM Sigplan Notices*, vol. 45, no. 6. ACM, 2010, pp. 86–97.

[23] "Nvidia gpudirect," Apr 2017, access date: 08.01.18 15:45:57. [Online]. Available: https://developer.nvidia.com/gpudirect