Deep Machine Learning on GPUs

Daniel Schlegel
University of Heidelberg, ZITI
schlegel@stud.uni-heidelberg.de

***Abstract*—With the increasing use of GPUs in science and the resulting computational power, tasks which were too complex a few years back can now be realized and executed in a reasonable time. Deep Machine learning is one of these tasks.**

In this paper the authors will provide a brief introduction to Neural Networks and the tools used to describe them. Then they will deduce the benefits and issues of running Neural Networks on GPUs.

***Index Terms*—Deep Machine Learning, Artificial Intelligence, GPU Computing.**

I. Introduction

SINCE their introduction in 1989 by Y. LeCun et. al. [11] Neural Networks (NNs) have performed well in a lot of different tasks. Starting with handwritten digit recognition to object classification and natural language processing, NNs provide a unified way to implement Machine Learning (ML) algorithms. Based on the high computational intensity and the high amount of parallelism, which can be exploited, they perform well on GPUs.

Furthermore, we entered the era of *big data* where thousands of GBs are generated each second. To make sense of all this data we depend on ways to process the data automatically and efficiently [13]. And GPUs are an important part of this progress because based on recent research, they are the most promising way to achieve the needed performance.

This paper aims to give a brief introduction to NNs, their history and the implementation of Deep Neural Networks (DNNs) on GPUs. The structure of the paper is as follows: In Section I the origins of machine learning and the application areas are explained. The next section deals with NNs in general. Therefore the functioning of NNs is explained and illustrated with a short example. Furthermore, tools, which can be used to implement NNs, are shown. Section III covers DNNs on GPUs starting with a brief introduction to GPU computing. This is followed by a performance evaluation and scalability considerations. The section is completed with an example. The paper is then concluded with an outlook and the conclusion.

A. What is learning?

The basis of Machine Learning is the human/animal learning. This is defined as every active, effort demanding (mental and psychomotorical), confrontation of a human with any objects of experience. In doing so intern representations are created and modified which causes a relative and permanent change of skills and capabilities [2]. Simply put, learning is when you do something often, you get better in it.

A simple and yet a clear example of learning is the behavior of rats. When rats encounter food with a novel smell or look, they try a bit at first. If they experience illness afterwards they associate this with the new food, and accordingly, they will not eat it again [18].

B. What is Machine Learning

The origin of ML lies in the area of Artificial Intelligence (AI). AI tries to transfer the human mind into a machine to enable machines make "intelligent" decisions like we do. ML is a sub-category.

ML is the attempt to imitate the human learning process. Based on the input these systems make decisions, that do not rely on explicitly defined functions. This means, that a system learns from data by itself, it evolves. For these ML systems, unlike for the human brain, it doesn't matter in which order (easy to hard) the data is presented. And thus with ML you can train a system to recognize words before it is able to recognize letters.

Flach [6] describes it that way: "Machine Learning is all about using the right features to build the right models that achieve the right tasks". This quote implies, that the design of a ML system is a sophisticated task which requires several iterations.

C. Application Areas

ML is used in many application areas. The most common used-cases are image processing and speech recognition.

But there is also ML in places that are not that obvious. One of the most prominent example is the Spam filter used in nearly every e-mail client. It has some predefined patterns and adapts itself to the needs of each users.

Another prominent and yet unknown example of ML is Google Translate. This is called Machine Translation and does not require a dictionary, it uses the user feedback to improve the translations [4].

II. Neural Networks

Neural networks are systems for information processing inspired by biological nervous systems like brains. They consist of numerous connected elements (neurons) working together to solve specific problems. Like people, NNs learn by example [20]. There are two main types learning in NNs, supervised and unsupervised. The difference between these types is, that the input data for supervised NNs is labeled, the one for unsupervised is not. So, supervised is

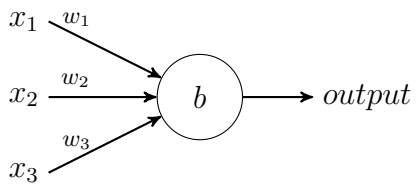

Figure 1: A simple neuron

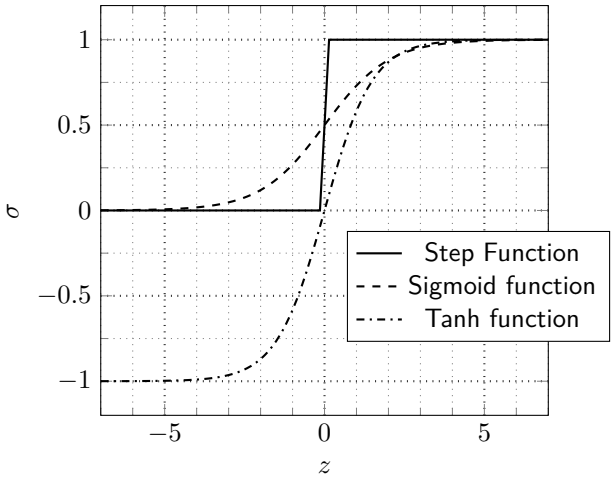

Figure 2: The step, Sigmoid and Tanh function

better if the task is known, handwritten digit recognition. Unsupervised can be used to find patterns in large set of unlabeled data. Since the most common case are supervised NNs, only these are covered in this paper.

During the training the NN is fed with thousands of classified objects, the training data. Each object is processed in one iteration. The amount of training of NNs is measured in epochs. In an epoch one object of each class the NN has to recognize is processed. To fully train a NN several thousand of epochs are needed.

Via backpropagation, the actual "learning" algorithm of the NN, the parameters of each neuron is adjusted to deliver the delivered value at the output [14]. This means the NN gets "smarter" the more data it learns from.

Furthermore, if the network is trained too well, overfitting can occur. That means, that the NN adapted itself perfectly for the training data an is not able to recognize objects, which were not in the training set. There are several ways to prevent the NN from overfitting e.g. the randomized deactivation of neurons during the training phase [19].

NNs are used to classify data. One of the most common examples are handwritten digit recognition, object recognition in images and natural language processing.

The latest implementations of NNs are Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs). CNNs are a subset of DNNs with a reduced amount of synapses. This results in lower computational intensity, but also adds a shift invariance during the feature extraction. Easy to say, if a mouth, a nose and two eyes are approximately in the right area it is detected as a face. This also means, that it is not possible to determine whose face it is.

A. How do Neural Networks work?

Like the human brain NNs use neurons. A neuron is a simple element which takes the inputs $x_i \in 1 \ldots n$ and combines them to the output. Figure 1 shows a neuron. There are several versions of neurons which compute the output in different ways.

If the output is computed by a simple step function (see equation 1) the neuron is called a perceptron. This is the "ancient" version because the transitions are not smooth and thus the outputs are either zero or one and do not represent the probability.

The use of the Sigmoid function (see equation 2 and figure 2) or the Tanh function as the transfer function is the up-to-date version. The Sigmoid and Tanh neurons provide a smoother transition between zero and one which is more desirable than the "hard" switching of the step function. For the classification, the assignment to a class, it is useful when the output represents the probability.

An example for the benefit is an image which contains several objects. When using a step function the probabilities are equal, or non existent, no matter how big the objects in the image are. If the Sigmoid function is used the probabilities are scaled according to the size or striking features. Therefore all recent NNs only use Sigmoid neurons.

The difference between this two activation functions is, that the Sigmoid function has a bias value. That means it can be shifted on the z-axis. This is especially important when the input data is not pre-processed (normalized and filtered). The Tanh function is used for the pre-processed data. It has no bias value and lies between -1 and 1. This behavior causes a better and faster learning when combined with pre-processed data [12].

$$x = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \leq 0 \\ 1 & \text{if } \sum_j w_j x_j + b > 0 \end{cases} \quad (1)$$

$$\sigma = \frac{1}{1 + e^{\sum_j w_j x_j - b}} \quad (2)$$

Equations 3 and 4 show the optimization algorithm of the backpropagation. The backpropagation implements the actual learning into the NN. During the training phase of the network the output is analyzed and the optimizer tries to adjust the weights and the biases of the neurons in a way to minimize the error of the output, thus minimizing the cost function. The training data set is separated into batches with a given size (a common size is 64 or 128). After each batch the backpropagation optimizes the parameters of the NN. The batch size affects the performance of the NN dramatically because the optimization is the most complex task in the training of the NN.

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k} \quad (3)$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l} \quad (4)$$

The cost function (equation 5) depends on the weights and the biases of all neurons in the network. It computes the error of the network $C(w, b)$.

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 \quad (5)$$

There are a lot of different functions used in a NN. For the basic understanding only a few are needed, which are explained in the following section.

Convolution The convolution is a mathematical operation that is often used to filter signals and images. In DNNs the convolution is mostly used as an edge detector and feature extractor. Sometimes the filter kernel is moved with a stride (it skips n pixels). This reduces the amount of computations and the output size.

Pooling Pooling or subsampling is used between different layers. It takes the output of several neurons and combines them to one output. A common form is the MaxPooling where only the biggest value of the grout is passed to the next layer. This reduces the amount of neurons and thus the computational intensity.

SoftMax The SoftMax function is placed right before the output. It is used to scale the probabilities (the outputs of the network, each output represents one class) between zero and one, also all probabilities sum up to one. For example in a NN that is trained to recognize digits and letters. If the input shows a "l" the network recognizes it as an "L" and "1", so the probabilities are (as an example) 90 % each which is 180 % in total. The task of the SoftMax function is to scale the outputs to 50 % each.

B. Example

A simple example for a NN is the handwritten digit recognition. It is inspired by Nielsen [20]. Picture 3 shows a simplified Directed Analytical Graph (DAG) for this network. The input images are taken from the MNIST[1] dataset, which provides centered and size-normalized digits. The size of all images is 28 × 28 pixels.

Based on the size of the input images the number of neutrons in the first layer, the input layer, is set to 28 × 28 = 784. The number of neutrons in the second layer, the hidden layer, is set to 16 and ten in the third layer, the output layer. If the input image shows the digit one the first output should be active and so on. Since a Deep Neural Network has to have, due to its definition, at least two hidden layers this isn't a DNN, but one hidden layer is sufficient to achieve an accuracy of 95 % in this task.

[1]http://yann.lecun.com/exdb/mnist/

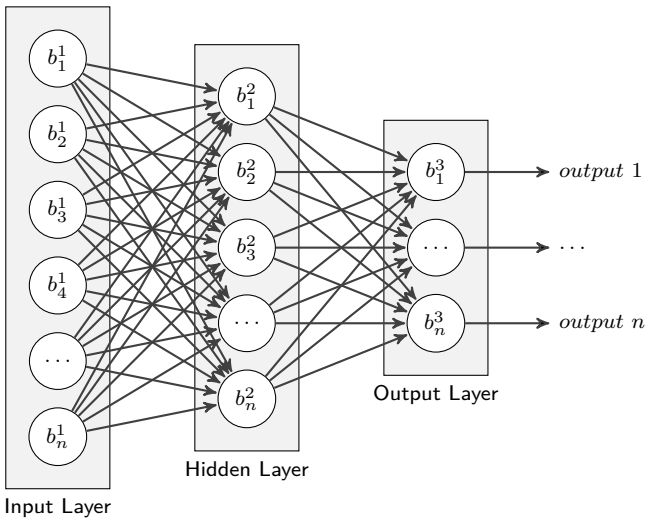

Figure 3: A DAG for handwritten digit recognition

C. Tools for Neural Networks

To simplify the description and implementation on NNs there are several frameworks and libraries implementing the functions needed in a NN. Since there are tools for different application areas and programming languages, but our focus is the GPU based implementation, we will present a very common framework with both, CPU and GPU, support and a recently released library for the GPU based computation of DNNs. This is, by far not a complete list. A summary of available frameworks can be found in [8].

1) Caffe: Convolutional Architecture for Fast Feature Embedding (Caffe) is the most popular framework for the creation on DNNs. It is open-source and available on GitHub[2]. The structure of the network is specified with a text based protocol, which enables fast editing of the network structure. Furthermore, all functions are implemented in C++ and CUDA-C, which enables a simple switching between CPU and GPU execution, even during the execution [8].

Listing 1 shows the configuration of a simple convolutional layer. In Caffe the layers are ordered from bottom to top, so the input layer is `data`. The layer has 96 filter kernels with a size of 11 × 11, which are initialized with a Gaussian function. The biases of the neurons, which bypass the convolution filter, are initialized with a constant value of zero, thus they are not used.

2) cuDNN: cuDNN is a library for DNNs which implements optimized functions for NVidia GPUs. It includes the most used functions like convolution, pooling, SoftMax and neurons with different activation functions. It is available for free at NVidia[3]

As shown in figure 4 there is a huge performance gap between the CPU and GPU implementation. But also, by using Caffe with cuDNN, a performance gain of about 27 %

[2]https://www.github.com/BVLC/caffe
[3]https://developer.nvidia.com/cuDNN

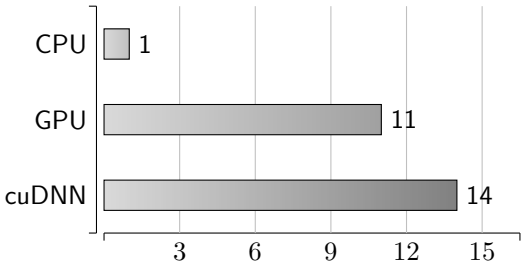

Figure 4: Performance comparision of Caffe

can be achieved. cuDNN is already included in the unstable branch of Caffe, the official and stable release is outstanding (at the time this paper was written).

3) cuda-convnet2: cuda-convnet2 is a small library written in C++ and CUDA-C. It was developed under the lead of Krizhevsky [9] for the ILSVRC-2012 (see section III) and is available at Google Code[4]. Like in Caffe the structure of the network is defined by a configuration file.

The special feature of this library is the possibility to run the network on multiple GPUs. There are several approaches of parallelism described by Krizhevsky [10]. By parallelizing the NN on eight GPUs a speedup of 6.25 can be achieved.

III. DML on GPUs

In 2010 the first GPU-based system entered the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The task in the ILSVRC is to train the system with a given set of images which are separated in 1000 classes. In the following step the system has to classify test images to the correct classes and mark position of the recognized objects. The winner of the challenge is the system with the smallest error rate. The previously mentioned system achieved an error rate of about 17 %. This result is nearly 10 % better than the second best system, which was CPU-based [17].

Another great example of DNNs on GPUs is NVidias answer to *Google Brain*. Google Brain is an unsupervised system used to find the two most common images in the Internet (they are cats and humans). Google implementation used 1000 nodes and took one week to find the answer. NVidia realized the same NN on a three node system with three GPUs each. Additionally, to the lower costs of the new system, the energy consumption is 100 times lower than the original one which is a very important factor in the near future.

A. GPU

GPUs are highly specialized processors for image processing. Starting in the 21st century the area of General-purpose GPU (GPGPU) Computing started to grow and is now an essential part of many scientific computing applications.

[4]https://code.google.com/p/cuda-convnet2/

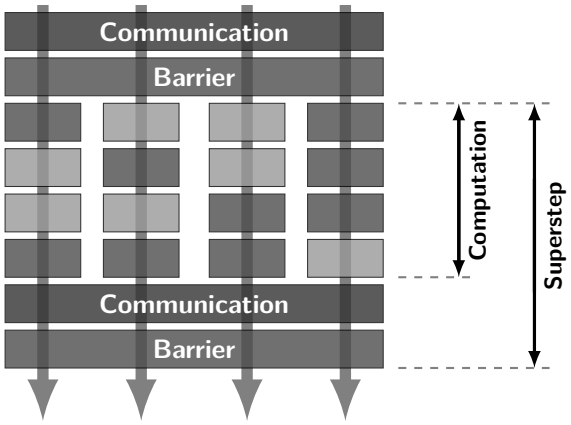

Figure 6: Illustration of the BSP Model [7]

1) Basic Architecture: The basic architecture of a GPU differs a lot from a CPU. The GPU is optimized for a high computational power and a high throughput. Both are needed for the graphics processing. The basic building block of a GPU is the *Streaming Multiprocessor (SM)*. The SMs consist of many ALUs, also called CUDA-Cores in NVidia GPUs. Each SM can run one warp (a bundle of 32 threads) at a time. Each thread in a warp performs the same operation. NVidia calls this model Single Instruction Multiple Threads (SIMT).

The full on-board memory bandwidth can only be utilized when the warps access the memory coalesced, that means, every thread accesses the memory location with its thread-id as offset. Furthermore the SMs are able to switch threads each clock cycle. Since the GPU can have a lot of outstanding loads and is able to switch the context every clock cycle it does not depend on the memory latency, like the CPU, but on the memory bandwidth.

2) Execution model: The execution model of a GPU can be described with the Bulk Synchronous Parallel (BSP) Model introduced by Valiant in 1990 [22]. Figure 6 shows an illustration of it. The program is split into supersteps. Each superstep has three phases, the computation, communication and the synchronization. Additionally Valiant proposes to start many more threads than physical resources are available to overcome the parallel slackness.

The kernels are executed in different blocks, each block consists of a given number of threads. Both, the number blocks and threads, can be indexed in three dimensions. This makes it easy to compute multidimensional problems.

B. Performance evaluation

The computation of DNNs is a task that fits excellent on a GPU. There is a large amount of parallelism that can be utilized. The most common kernels are matrix multiply, convolution and functions with no data dependencies at all (see neurons).

The achieved performance highly depend on the implementation of the NN. When using cuDNN a performance of nearly 2.5 TFLOPS per layer on a NVidia GTX 980 are

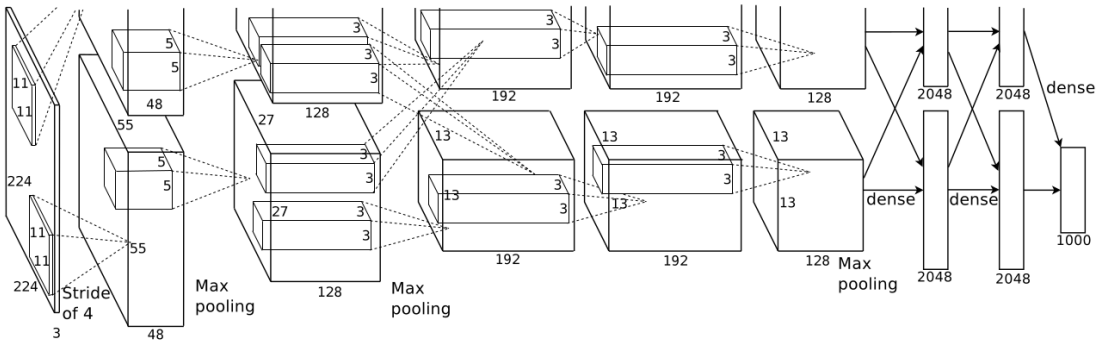

Figure 5: DAG of the winner of the ImageNet Challenge 2012 [9]

Parallel loading	Data parallelism		Caffe	Caffe with cuDNN
	2 GPUs	1 GPU		
Yes	23.39	39.72	26.26	20.25
No	28.92	49.11		

Table I: Comparison of training times [5]

possible. This corresponds to 51 % of the peak performance [3].

C. Scalability

Currently there are only a few systems using DNNs split over several nodes and there is no (published) effective way for the communication of the split NN yet. Most current systems do not split the NN itself into several parts for each node. Each node is running independently with the full network. Only the backpropagation path is shared between the whole system. After each batch the deltas of the new weights and biases are collected, averaged and send to each node.

In contrast to the cluster approach there are several approaches to implement DNNs on a multi GPU system. Some of these systems were proposed by Ding et al. [5], Yadan et al. [24], Zou et al. [25], Paine et al. [16] and Krizhevsky [9].

Ding [5] implemented both, a single GPU and multi GPU solution with the Python based framework *THEANO*. They achieved a speedup of 1.7 with their implementation (see table I). Although it is still 15 % slower than the single GPU implementation with Caffe they rate it as a success because they used a full python implementation, which is very flexible. In their paper they also point out, that they were limited to two GPUs because in order to have peer-to-peer access, the cards have to be connected to the same PCIe root complex.

Another reason to use a multi GPU system is the limited amount of on-board memory. Recent GPU generations have a maximum of 12 GB of memory. Therefore, by doubling the number of GPUs, also the memory is doubled and thus the network and the training batch size can be increased.

D. Example

The previously mentioned system (see figure 5) uses two NVidia GTX 580 General Purpose GPUs (GPGPUs) and a CNN, which was optimized for GPU-to-GPU communication (see figure 5). The upper part of the DAG is running on the first GPU and the lower part on the second one.

The system uses seven hidden layers with three additional MaxPooling layers between the convolutional layers. The complete amount of neurons used in the hidden layers is 650,000 with 60,000,000 weights and biases. The output is a SoftMax layer with 1000 outputs. This layer is executed on one GPU since it needs a lot of communication and less computation. To prevent the system from overfitting they used the dropout approach proposed by Srivastava et al. [19]. The program was further optimized by not using all-to-all communication between all layers. Only between layers two and three and five, six and seven.

IV. Outlook

Since there is a lot of research on multi GPU solutions for DNNs with promising results (see section III-C), it can be expected that there will be several multi node solutions in the near future.

One of the biggest issues, the limited on-board memory of the GPU, should be solved in the near future. In the last years the memory size has nearly quadrupled.

The biggest challenge, so far, is to find ways to scale the networks on cluster level, thus finding ways to communicate efficiently. But multi GPU systems, like the one proposed by Krizhevsky [9] show that, by using special CNNs, the communication can be reduced without affecting the accuracy of the network. This results show, that it is possible, but the mapping and communication has to be adapted to these new architectures.

V. Conclusion

DNNs offer an easy and unified way to implement ML. By using frameworks it is fairly easy, compared to a custom program, to implement them with a good performance. The basic functions of the DNNs are applicable to all kinds of input data, which makes the use of these networks very desirable.

And since there is a high amount of parallelism that can be exploited with kernels that map well to a GPU. Based on this it can be said, that the role of GPUs in the field of ML will increase over the next years.

Appendix A
Code Listings

Listing 1: Concolution Layer in Caffe

```
# Simple Convolutional Layer
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 96
    kernel_size: 11
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
```

References

[1] E. Alpaydin. *Introduction to Machine Learning.* Adaptive Computation and Machine Learning Series, MIT Press. 2014.

[2] Lotter, Hempel. *Lernen, Lernschwierigkeiten – Diagnostik der Lernvoraussetzungen.* Regierung Oberbayern. 2008.

[3] S. Chetlur. *cuDNN: Efficient Primitives for Deep Learning.* arXiv preprint arXiv:1410.0759c2. 2014

[4] T. Brants et. al.. *Large Language Models in Machine Translation.* EMNLP-CoNLL. 2007.

[5] W. Ding, et. al.. "Theano-based Large-Scale Visual Recognition with Multiple GPUs" ICLR. 2015.

[6] P. Flach. *Machine Learning: The Art and Science of Algorithms that Make Senso of Data.* Cambridge University Press. 2012.

[7] H. Fröning. *GPU Computing Slides* University of Heidelberg, ZITI. 2014.

[8] Y. Jia et. al.. *Caffe: Convolutional Architecture for Fast Feature Embedding.* arXiv preprint arXiv:1408.5093v1, 2014.

[9] A. Krizhevsky, I. Sutskever and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks.* NIPS. 2012.

[10] A. Krizhevsky. *One weird trick for parallelizing convolutional neural networks.* arXiv:1404.5997 [cs.NE]. 2014.

[11] Y. LeCun, et. al.. *Backpropagation applied to handwritten zip code recognition* AT&T Bell Laboratories. 1989.

[12] Y. LeCun, et. al.. *Efficient BackProp* Neural Networks: Tricks of the trade, Springer. 1998.

[13] K. P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press. 2012.

[14] M. A. Nielsen, *Neural Networks and Deep Learning.* Determination Press. 2015.

[15] NVidia. *User Guide – cuDNN Library* NVidia. DU-06702-001_v6.5. 2014.

[16] T. Paine et al.. *Gpu asynchronous stochastic gradient descent to speed up neural network training.* arXiv preprint arXiv:1312.6186. 2013.

[17] O. Russakovsky el. al.. *ImageNet Large Scale Visual Recognition Challenge.* arXiv preprint arXiv:1409.0575. 2014.

[18] S. Shalev-Shwartz, S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press. 2014.

[19] N. Srivastava, et. al.. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* University of Toronto, Department of Computer Science. 2014.

[20] C. Stergiou, D. Siganos. *Neural Networks.* Imperial College London. http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#WhatisaNeuralNetwork, last visited 28.12.14

[21] X. Tang. *Introduction to General Purpose GPU Computing* University of Rochester. 2011.

[22] L. G. Valiant. *A bridging model for parallel computation.* Communications of the ACM, Volume 33 Issue 8. 1990.

[23] J. Wart, et. al.. *Efficient mapping of the training of Convolutional Neural Networks to a CUDA-based cluster* Eindhoven University of Technology, The Netherlands. 2011.

[24] O. Yadan et. al.. *Multi-gpu training of convnets.* arXiv preprint arXiv:1312.5853. 2013.

[25] Y. Zou et. al.. *deep learning platform and its applications.* Proceedings of the VLDB Endowment. 2014.