Deep Learning and High Performance Computing

Mohamed Ez zaouia
Research Master at Blaise Pascal University - ISIMA
ezzaouia.mohamed@gmail.com
Clermont Ferrand - France

***Abstract*—Neural network and especially deep learning are a high active research filed in machine learning, currently provide the best solutions and results to many problems in computer vision (image [1], [2], speech [3], face [4], emotion [5]), NLP (natural language processing [6]), even for fraud detection, example of by PayPal which currently use deep learning for electronics credit cards fraud detection. On the other hand the large amount of data in our days oblige us to scale our model in order to improve the efficiency of the model. In this field, HPC (High Performance Computing) play a major role and brings a big enhancements and good results. In this survey we provide a brief introduction to deep learning and we highlight some of the current research efforts and HPC parallelism approaches used in the field of deep learning.**

Keywords-component; Machine Learning; Neural Network; Deep Learning; HPC; Parallelism; Computer Vision.

I. INTRODUCTION

Every one of us may already use a GPU cluster on the cloud by using some smartphone applications like *Shazam* for music recognition or by using some research engines like Google or even some APIs like Microsoft Face APIs all of this shows how HPC touches our daily life by powering a huge and massive real-time computing back-ends.

Data is growing exponentially in its tree 3V dimensions Volume, Velocity and Variety with (regarding to domo.com statistics) every minute of the day users send more than 200 million email, Google receive more than 4 million query, more than 72 hours of new videos uploaded on YouTube, and around 2.5 million peaces of content are shared on Facebook. In fact, that is very difficult and sometimes impossible de handle, manage and analyze this amount of data using traditional software and technologies. This exponential growth in data brings serious transformations and potential in many sectors such as enterprise, finance, healthcare, services, energy, commerce [7] etc. and also brings big challenge to the scientific research area in order to handle this amount of data with new smart methods and technologies.

While big amount of data brings more opportunities to transform our industry and hence our society, collecting useful and precious may be a big deal and a difficult task. Rapidly growing amount of data require many investment in term of development in advanced multidisciplinary methods and technologies. Nowadays, machine learning approaches and algorithms emerges to play a major role by providing good solutions and results to many problems especially the ones dealing with big data [8], they are hugely used in many fields, industries and companies to provide predictions and analysis from data. Deep learning is one promising branch of machine learning especially in data analytics, in fact it can magically automate the hard task in machine learning design which is extraction of features in different level of abstraction, it has radically improved the computer's ability to recognize, detect and analyze data such as human do [9], currently the main focus is on specific perceptual task with big success. As mentioned before deep learning differ from conventional learning approaches by automatically extract representations (features) [10]. Industry giants Amazon, EBay, Google, Apple Baidu, Netflix, Facebook, and Microsoft are working with the most players in this field to improve their products powered by deep learning.

In fact, the data is keep growing and so deep learning is emerging as major player with a key role in big data prediction [8], especially with the advanced improvement in technology (software and hardware) for instance GPUs [11] which accelerate massively processing and computing. The main goal of this paper is neither not to present a comprehensive survey of all deep learning related approaches nor big data, but to present the most important methods of deep learning coupled with HPC. The rest of this paper is organized as follows. Section 2 presents a brief overview of the main two approaches commonly used for deep learning and we conclude with some current progress in large scale deep learning. As note, we use the terms machine(s), node(s), CPU(s), GPU(s), and Thread(s) interchangeably in the context of this paper.

II. DEEP LEARNING

Deep learning can be described as a set of techniques for learning in neural networks which is machine learning approaches inspired by the human brain: They are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity see [12] for more details. The current section will highlight the two well-known deep learning architectures: Deep Belief Network (DBNs) [13] and Convolutional Neural Networks (CNNs) [14].

A. *Deep Belief Network*

Deep Belief Networks (DBN's) are probabilistic generative models that contain many layers of hidden variables, in which each layer captures high-order correlations between the activities of hidden features in the layer below. The top two layers of the DBN form an undirected bipartite graph with the lower layers forming a directed sigmoid (example of activation function) belief network, as shown in Fig. 2.1.

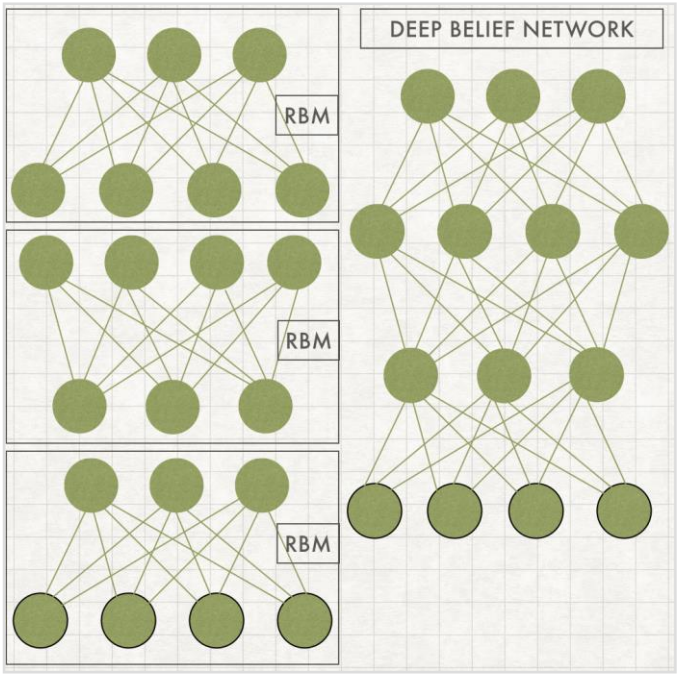

Figure 2.1. Left: Greedy learning a stack of RBM's in which the samples from the lower-level RBM are used as the data for training the next RBM. Right: The corresponding Deep Belief Network.

G. E. Hinton [11] introduced a fast, unsupervised learning algorithm for these deep networks. A key feature of this algorithm is its greedy layer-by-layer training that can be repeated several times to learn a deep, hierarchical model. The learning procedure also provides an efficient way of performing approximate inference, which only requires a single bottom-up pass to infer the values of the top-level hidden variables.

The main building block of a DBN is a bipartite undirected graphical model called the Restricted Boltzmann Machine (RBM). The goal for training a DBN is to learn the weight and biases between layers, which is conducted by an unsupervised learning of the main blocks of a DBN which are RBMs. As shown in Fig. 2.2 the two first layers (the input layer and first hidden layer) form the first RBM. The bipartite structure of the graph allows the training of weights W of each RBMs using Gibbs sampling [11].

It is commonly used to perform a layer-by-layer unsupervised pre-training of the RBMs constituting the DBN, this process allows to avoid local optimum and over-fitting the model. We should mention the efficiency of the algorithm from time complexity point of view which is linear to the size of RBMs (and number of RBMs also). Each layer in the DBN is a high level abstraction of the information of the data structure.

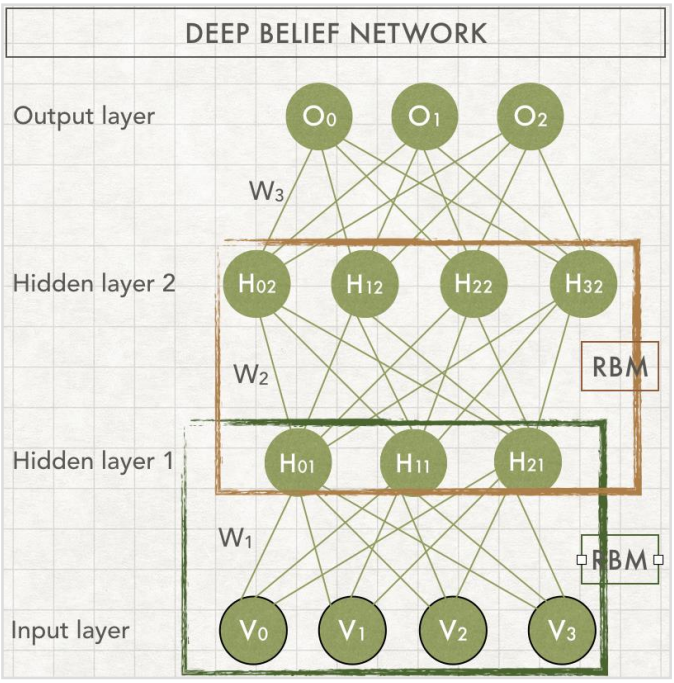

Figure 2.2. A BDN which consists of two hidden layers, first one with three neurons and second one with four neurons; one input later, one output layer with four neurons and tree neurons respectively. Any two adjacent layers can form a RBM. The outputs of current RBM (for instance H_{j1} in the first RBM) are the inputs of the next RBM (H_{j2} in the second RBM). The weights W can then be fine-tuned with labeled data after pre-training.

The reader may refer to [15] for information about RBMs with Bernoulli distribution used for input layer and hidden layers. We should mention that there are others alternatives for pre-training approaches rather than RBMs such as [16] and [17].

To summarize, as mentioned before DBN is based on a greedy algorithm for layer-by-layer pre-training, the algorithm is efficient in terms of time complexity, a back-propagation algorithm is also used for well tuning the model which increase the accuracy of the model.

B. *Convolutional neural networks*

Convolutional Neural Networks (CNNs) are very similar to ordinary Neural Networks, CNNs architectures benefit from the fact of assuming that the inputs of the model are explicitly images, this increase the efficacy of training the model and hugely reduce the amount of the parameters in the model, by encoding certain properties into the architecture related to the fixed input type (as image).

By way of comparison, Regular Neural Networks (RNNs) receive an input (vector), and transform it through a series of transformations (hidden layers). Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer; neurons in a each layer operate

fully independently without any shared connections. The last fully-connected layer is the output layer and in it represents the prediction class scores. One of the big issue of a such architecture is scalability, in fact, RNNs does not scale up well, for instance, an image of size 32x32x3 (32 wide, 32 high, 3 color channels), input layer would have 25*25*3 = 1875 weights which is manageable, but does not scale to big image, for example 300*300*3 = 270000 weights in the some way the parameters would hugely increase which would led to over-fitting. To overcame this issue, CNNs by using the fact that the input is image and hence they build the architecture in different fashion, in particular neurons are arranged in three dimensions: width, height, depth (depth refers to the third dimension of an activation volume, not to the depth of a full NN, which can refer to the total number of layers in a network.), for instance 25*25*3 for width, height, depth respectively, the most important fact of CNN is that neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected fashion [18].

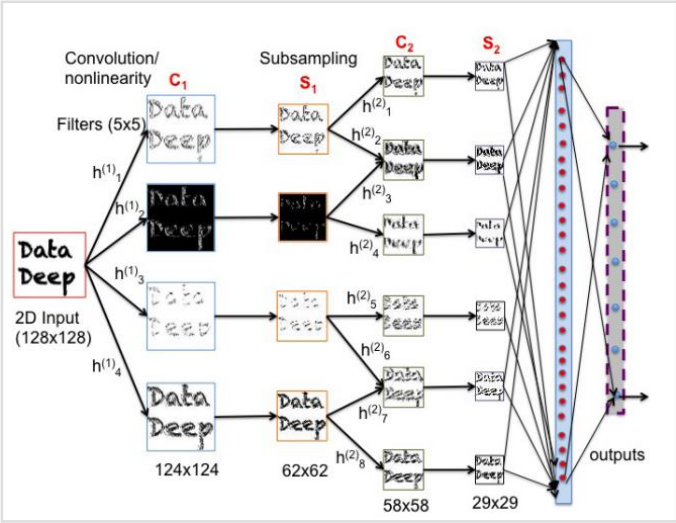

Figure 2.3. Left: Illustration of a typical convolutional neural network architecture. The input is a 2D image, which convolves with four different filters (i.e., $h^{(1)}_i$, i = 1 to 4), followed by a nonlinear activation, to form the four feature maps in the second layer (C1). These feature maps are down-sampled by a factor of 2 to create the feature maps in layer S1. The sequence of convolution/nonlinear activation subsampling can be repeated many times. In this example, to form the feature maps in layer C2, we use eight different filters (i.e $h^{(2)}_i$, i = 1 to 8): the first, third, fourth, and sixth feature maps in layer C2 are defined by one corresponding feature map in layer S1, each convoluting with a different filter; and the second and fifth maps in layer C2 are formed by two maps in S1 convoluting with two different filters. The last layer is an output layer to form a fully connected 1D neural network, i.e., the 2D outputs from the last subsampling later (S2) will be concatenated into one long input vector with each neuron fully connected with all the neurons in. the next layer (a hidden layer in this figure).

More general, a typical CNN is a collection of many layers of hierarchy with some layers for feature representations and others as a type of CNNs for classification [19]. It often starts with two altering types of layers called convolutional and subsampling layers: convolutional which is a mathematical term, defined as applying a function repeatedly across the output of another function layers in the context of deep learning means perform convolution operations with several filter maps of equal size at all possible offsets, while subsampling also called down-sampling refers reducing the sizes of proceeding layers by averaging pixels within a small neighborhood (or by max-pooling [20]). Fig. 2.3 shows a typical architecture of CNNs. The input is first convoluted with a set of filters (C layers in Fig. 2.3). These 2D filtered data are called feature maps. After a nonlinear transformation, a subsampling is further performed to reduce the dimensionality (S layers in Fig. 2.3). The sequence of convolution, subsampling can be repeated many times.

To summarize, a CNN is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. This architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). CNNs are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

III. DEEP LEARNING AND HPC

Recently we have seen a lot of deep learning algorithms, in addition to algorithms, capability based on training a very large neural network is an important factor of improved results. In fact, it has been observed in machine learning that the biggest the model is the more accurate result will get, that is, scaling deep learning with the respect the training examples, the number of the model's parameters, or both, can greatly improve the prediction accuracy. In the other side, scaling a deep learning model can take much more time to train and tune the model. Therefore, HPC is major factor which can address this issues by using parallelism. Mainly, there are two ways to scale a neural network the first one is by using ***data parallelism*** which consist of using the same model for every machine (or Core, CPU, Thread) each one with different part of data, the second way is **model parallelism** in contrast, use the same data for each machine (Core, CPU, Thread) but split the model among machines (Cores, CPUs, Threads), in fact, a third way for parallelism need to be mentioned which is an hydride approach combining data and model parallelism. In this subsections, we will present parallel methods: Data parallelism and model parallelism which are introduced in Google DistBelief [21] for CPU clusters, and also used in Google COTS systems [22] for GPU servers as well as Facebook multi-GPU training [23] after that, we will present some work examples done using those methods.

A. Data Parallelism

Data parallelism launches multiple model replicas for different mini-batches, and collects gradients to update model parameters. Model parallelism involves multiple workers each holding parts of the model and swapping activations and errors to complete a mini-batch. Besides improving performance,

model parallelism reduces memory consumption for each worker, thus make it possible to build larger models. Data parallelism is widely used in multi-GPUs for easy to program.

The idea behind it is that, each machine has the same own copy of the model and a chunk of data from the splitted training set; this architecture can be classified under Flynn's architecture called SPMD for Single Program Multiple Data, as shown in Fig. 3.1.

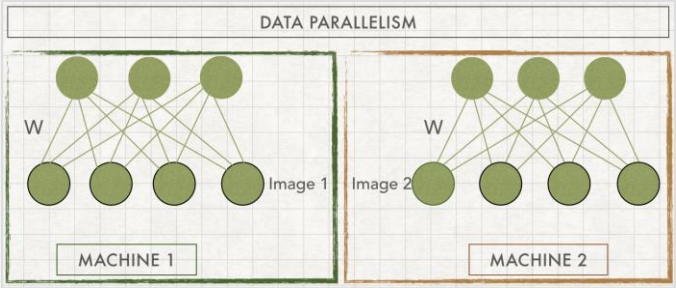

Figure 3.1. Illustration Data parallelism over images in a batch, which each machine host the entire copy of the model and the data is chucked over machines.

The main advantage of data parallelism is that each machine gets a copy of the model, allowing each one to function completely independently on its data without any inter-communication.

This is often desirable in cases where the model fits within the memory capacity of a single stack and the capacity overhead of replicating the model on all machine stacks is acceptable. Further, by increasing batch size, data parallelism can scale out to a large number of machines. For example, if there are 8 machines and the batch size is 256, then each machine gets 32 input images which may result in low CPU/GPU usage. If we increase the batch size to 1024, then each machine can get 128 input images and higher CPU/GPU usage. However, increasing the batch size can increase response time for latency for critical prediction tasks and adversely affect convergence rates in model training. Therefore, the batch sizes are typically set to be hundreds. For example, AlexNet [1] uses a batch size of 128 and VGG nets [1] use 256. Others issues of a such parallelism technique is the need to synchronize model across machines to average the result. Also data transfer over the network way adversely affect such based implementation, one last problem is the difficulty to fit big models on CPU/GPUs.

B. Model Parallelism

The second way to scale a neural network is to use Model parallelism, the idea behind this approach is the split the model on machines, for instance, in the case of the machines we give a half of neural network to the first machine and second half of neural network to the second machine and using the same data for both machines, with this architecture we can get high level of scalability with too much larger models but we end up needing more frequent synchronization, even though we end up moving data each time, adding to this the problem of the interconnection and the independence between the chunks of the model parallelized over multiple machines when some neuron one machine may need some input from another neuron in another to compute an output. Fig. 3.2. illustrate this model.

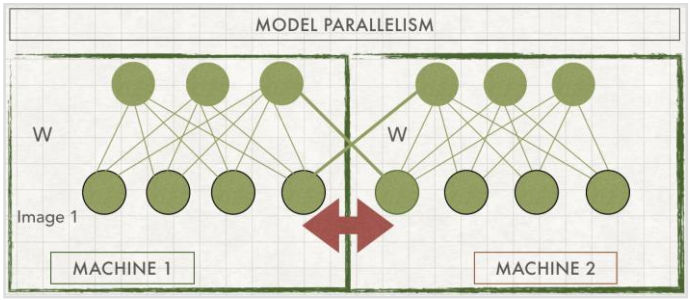

Figure 3.2. Illustration Model parallelism over neurons, which each machine host a part of the model and the entire model is working on the same data, neurons between machines share connection and synchronization is needed.

Some group of researchers [22] used model parallelism in MPI when MPI starts a single Linux process for each GPU in the cluster and pass messages between those programs running in each GPU so when the neural network is splitted on GPUs for instance two GPUs GPU1 and GPU2 we will have a connections that cross GPUs which need to share information and we need to orchestrate all the message passing to make sure that the shared values between neural network models in GPUs goes to each other, and this message passing is handled in the scope of neural network with a simple distributed array abstraction and all the communication for this architecture is hidden in this array.

The problem is that the infrastructure been used really does not scale well with the GPUs and it not so efficient with this large neural network.

One approach to overcome this problem is to use some hardware designed with HPC in mind like the use of Infiniband (IB) which is a computer-networking communications standard with very high throughput and very low latency (more than 50 Gbps, microsecond latency), for example [15] they used GTX 680 GPUs which can achieve 1 TFLOPS each for an ideal workload, combined with the software like MPI (Message Standard Interface), MVAPICH2 which is a GPU aware implementation of the standard MPI, at the time of the this writing MVAPICH2 support MPI 3.0 and deliver high performance, scalability and fault tolerance for HPC systems and servers using InfiniBand and enables message passing cross GPUs with MPI even if this GPUs are in different machines. While DistBelief [21] can learn with very large models (more than one billion parameters), its training requires 16,000 CPU cores, which are not commonly available for most researchers, Coates et al. COTS HPC [22] system trains a network with more than 11 billion parameters in only three days, which requires about 82 GB memory, clearly, such a model is too large to it into one single node (machine) using data parallelism, and thus needs to be partitioned using model parallelism. The COTS used by Coates et al. is a cluster of 16 GPU servers and as mentioned before with Infiniband adapter for interconnects and MPI for data exchange in a cluster. Each server is equipped with four NVIDIA GTX680 GPUs, each having 4GB of memory.

With well-balanced number of GPUs and CPUs, COTS HPC is capable of running very large-scale deep learning.

IV. Conclusion

In this paper we have briefly introduced deep learning by highlighting its two most important architectures: Deep Belief Network and Convolutional Neural Network. After that we have explained the main approaches for building a distributed deep learning: Model parallelism and Data parallelism, each approach come with its advantages and disadvantages. Also we mention that the current HPC oriented hardware increases hugely the efficiency and the speedup of the entire model, and this will be for sure a major player for large-scale distributed deep learning or more general machine learning algorithms.

References

1. G. E Hinton, NIPS. «ImageNet Classification with Deep Convolutional Neural Networks». 2012.
2. S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich «Going Deeper with Convolutions». 19-Sept-2014.
3. Il. Sutskever, O. Vinyals, Q. VV Le. «Sequence to sequence learning with neural networks» Advances in Neural Information Processing Systems. 2014.
4. O. M. Parkhi. A. Vedald, A. Zisserman. «Deep Face Recognition». 2015.
5. S. Eb. Kahou, X. Bouthillier, P. Lamblin. «EmoNets: Multimodal deep learning approaches for emotion recognition in video». 30-Mar-2015.
6. S. R. Bowman, G. Angeli, C. Potts, C. D. Manning. «A large annotated corpus for learning natural language inference. In Conference on Empirical Methods in Natural Language Processing». EMNLP 2015.
7. (2011, May). Big Data: The Next Frontier for Innovation, Competition, and Productivity. McKinsey Global Institute [Online]. Available: http://www.mckinsey.com/insights/business_technology/big_data_the_next_frontier_for_innovation
8. J. Lin and A. Kolcz at Twitter, Inc «Large-Scale Machine Learning at Twitter». 01-May-2012.
9. Deep learning applications and challenges in big data analytics [Online]. Available: https://www.vlab.org/2014/09/28/deep-learning-intelligence-from-big-data/
10. C. XUE-WEN, L. XIAOTONG «Big Data Deep Learning: Challenges and Perspectives». 20-April-2014.
11. [11] G. E. Hinton «A Fast Learning Algorithm for Deep Belief Nets». 17-May-2006.
12. Neural Networks and Deep Learning [Online]. Available: http://neuralnetworksanddeeplearning.com/
13. V. Nair, G. Hinton, «3D object recognition with deep belief nets». 2009.
14. P. Le Callet, C. Viard-Gaudin, and D. Barba, ‘‘A convolutional neural network approach for objective video quality assessment» Sep-2006.
15. Restricted Boltzmann Machines (RBM) [Online]. Available: http://deeplearning.net/tutorial/rbm.html
16. H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, «Exploring strategies for training deep neural networks». 2009.
17. H. Lee, A. Battle, R. Raina, and A. Ng, «Efficient sparse coding algorithms». 2006.
18. Convolutional Neural Networks [Online]. Available: http://cs231n.github.io/convolutional-networks/
19. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, «Gradient-based learning applied to document recognition». 1998.
20. D. Scherer, A. Müller, S. Behnke, «Evaluation of pooling operations in convolutional architectures for object recognition». 2010.
21. Dean, J., Corrado, G.S., Monga, R., et al, Ng, A. Y. «Large Scale Distributed Deep Networks. In Proceedings of the Neural Information Processing Systems (NIPS’12)». 2013.
22. Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y. «Deep learning with COTS HPC systems. In Proceedings of the 30th International Conference on Machine Learning (ICML’13)». 2013.
23. Yadan, O., Adams, K., Taigman, Y., Ranzato, M. A. «MultiGPU Training of ConvNets». February-2014.