Chapter 5

Overcoming Challenges in Predictive Modeling of Laser-Plasma Interaction Scenarios. The Sinuous Route from Advanced Machine Learning to Deep Learning

Andreea Mihailescu

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.72844

Abstract

The interaction of ultrashort and intense laser pulses with solid targets and dense plasmas is a rapidly developing area of physics, this being mostly due to the significant advancements in laser technology. There is, thus, a growing interest in diagnosing as accurately as possible the numerous phenomena related to the absorption and reflection of laser radiation. At the same time, envisaged experiments are in high demand of increased accuracy simulation software. As laser-plasma interaction modelings are experiencing a transition from computationally-intensive to data-intensive problems, traditional codes employed so far are starting to show their limitations. It is in this context that predictive modelings of laser-plasma interaction experiments are bound to reshape the definition of simulation software. This chapter focuses an entire class of predictive systems incorporating big data, advanced machine learning algorithms and deep learning, with improved accuracy and speed. Making use of terabytes of already available information (literature as well as simulation and experimental data) these systems enable the discovery and understanding of various physical phenomena occurring during interaction, hence allowing researchers to set up controlled experiments at optimal parameters. A comparative discussion in terms of challenges, advantages, bottlenecks, performances and suitability of laser-plasma interaction predictive systems is ultimately provided.

Keywords: predictive modeling, machine learning, deep learning, big data, cloud computing, laser-plasma interaction modeling

1. Introduction to laser-plasma interaction simulations

Numerous significant technological advancements mark the nearly six decades that have elapsed since the invention of the laser. We are nowadays facing a dramatic increase in terms of attainable laser powers and intensities concomitantly with a drastic shortening of pulses

duration. Super-intense lasers such as HERCULES [1], TPL [2], Vulcan [3] and Astra Gemini [4] or PHELIX [5] constitute a notable achievement in terms of chirped pulses intensity: an increase by six orders of magnitude within less than 10 years. Next generation 10-PW laser systems are currently under consideration in various laboratories around the world. To resume to just one example, the 10-PW ILE APOLLON [6] is envisaged to deliver an energy of 150 J in 15 fs at the last stage of amplification after the front end, with a repetition rate of one shot per minute, its intensity being expected to reach 10^{24} W·cm^{-2}. Such elevated intensities are the foregoers of the so called ultrarelativistic regime applications, a regime in which not only the electrons but also the ions become relativistic within one laser period. As matter under extreme conditions can now be relatively easily generated and investigated, we are witnessing a worldwide advent of laboratory research in totally "new physics", from ultrarelativistic laser plasmas to high-energy particle acceleration and generation of high-frequency radiation in the extreme-ultraviolet (XUV) and soft-X-ray regions. X-ray production by means of high intensity laser-plasma interaction experiments is of particular interest for the scientific community since this is a way of attaining increased brightness X-rays, with good coherence and consequently high quality sources of radiation. Among the variety of laser-based mechanisms deployed for this purpose, the most notable are betatron generation from laser wakefield acceleration [7] and high-order harmonics generation (HHG) [8].

In spite of the multitude of opportunities, there are still technological issues to be addressed and there are still numerous phenomena occurring during the interaction that are not yet fully understood. Some of these may be potentially damaging to experiments (e.g. hydrodynamic or parametric instabilities, hot electrons), hence their mitigation is vital. Ultimately, optimizing interaction conditions requires state-of-the-art theoretical and computational investigations.

In terms of simulation software, traditional approaches entail either hydrodynamic (fluid) or kinetic codes, in accordance with the laser-plasma interaction regime. Often, choosing between the two implies an inevitable dismissal of certain phenomena within reasonable accuracy limits. Modeling processes like particles' acceleration, plasma heating, parametric instabilities that occur during the interaction of ultrashort (pulse duration of sub-picoseconds down to tens of femtoseconds) and intense (intensity higher than 10^{17} W·cm^{-2}) laser pulses with plasma requires mainly a kinetic treatment and this is normally achieved through the Particle-In-Cell method (PIC) [9], the most reputed among the numerical tools employed in plasma physics and in laser-plasma interaction investigations. Albeit being recognized as a suitable approach for analyzing the highly transient physical processes in the non-linear regime associated with ultrafast laser energy coupling to matter, PIC based codes are subject to nonphysical behaviors such as statistical noise, non-physical instabilities, non-conservation, and numerical heating. Secondly, they require considerable computational resources, being far more demanding than the fluid ones that are normally deployed to study phenomena on a nanosecond scale with "coarser" accuracy. For instance, running a 1D PIC with a reasonable number of particles per cell, a fine grid and a small time resolution can claim up to more than 20 CPU hours on a single-processor PC for simulating what happens during a few femtoseconds of interaction. The distribution function at any given time, in a 3D3V PIC code is six dimensional in nature. Should 100 grid points be allocated for each dimension and representing each grid point in eight byte double precision, then, the system would require as far as 7 TB alone, just to store this data structure. In spite of the recent advent of computing technologies, running high

accuracy 3D or even 2D kinetic simulations is still a challenging task even if we are talking about a full migration towards the GPUs.

Various simplified codes have been hitherto been built and successfully used with reasonable compromises between accuracy on one hand and storage requirements and speed on the other. The LPIC++ [10, 11], XOOPIC [12] and PIConGPU [13] are some good examples in this sense. Restraining the number of dimensions, in conjunction either with object oriented programming, either with code parallelization, makes it possible to gain increased resolution (but over fewer dimensions) with less fancy hardware. Among the state-of-the-art PICs employed for simulating a variety of laser-plasma problems are the well-established EPOCH [14], VSim [15], OSIRIS [16–18], and QuickPIC [19, 20]. Fully relativistic, parallelized and multidimensional, they all incorporate additional features accounting for phenomena normally disregarded by traditional PIC methods, therefore moving the simulations closer to the real world. For example, EPOCH includes multiphoton, tunneling and collisional ionisations. The latter two can also be found in OSIRIS. VSim is a hybrid code (combining kinetic and hydrodynamic treatments), while OSHUN [21, 22] permits the user to introduce multiple ion species. At the same time, system resources can be spared by either reducing the number of dimensions (user option encountered in EPOCH) or by separating out the time scale of the evolution of the driver from the plasma evolution, thus transforming a fully 3D electromagnetic field solve and particle push into a sequence of 2D solves and pushes (QuickPIC's algorithm). Highly optimized to run even on a single CPU, these codes are scalable over a large number of cores, featuring the dynamic load balancing of the processors. Parallelization approaches include not only the MPI and Open MP but SIMD Vectorization, with most of these above mentioned simulation environments having CUDA enabled versions as well. Running a PIC code on top of the line GeForce or on Tesla can lead to significant improvements in terms of speed [23–32] while maintaining a fairly large number of particles per cell. Breakthroughs have been reported especially with the particle push [33–35] and particle weighing [36–38] algorithms but also with the parallelization during the current deposition phase [39, 40]. Successful attempts of integrating these schemes while trying to mitigate some of the factors known to limit GPU performance—communication overhead between GPU and CPU, memory latency versus bandwidth, the relatively low level of multitasking or I/O efficient management when reading and writing to files—count in Jasmine [41, 42] or FBPIC [43, 44].

As cloud, big data and AI based technologies are nowadays becoming pervasive in all the fields of the economy, predictive modeling should become just as ubiquitous in every research area, being a comfortable and reliable alternative for designing optimized experiments or for estimating potential results.

This chapter is presenting an overview of an entire class of predictive systems for laser-plasma interaction built at the National Institute for Lasers, Plasma and Radiation Physics—blending in big data, advanced machine learning algorithms and deep learning—with improved accuracy and speed. Making use of terabytes of already available information (literature as well as simulation and experimental data) such systems have the potential of revealing various physical phenomena occurring in certain situations, hence enabling researchers to set up controlled experiments at optimal parameters. Whilst the most obvious advantage of deploying predictive and/or prescriptive modeling is the considerably diminished running time in comparison to classic simulation codes, the motivation goes further than this, to having a readily compiled

report containing the most favorable interaction conditions or warnings on the imminent presence of destructive phenomena. However, efficiently extracting, interpreting, and learning from very large and heterogeneous datasets requires new generation scalable algorithms as well as new data management technologies and cloud computing. In this sense, a big step forward was the deployment of Hadoop [45], together with its MapReduce [46] algorithm and the Mahout library [47, 48]. Several other libraries were jointly used for deep learning purposes, namely Theano [49], TensorFlow [50], Keras [51] and Caffe [52]. Promising results—correctly predicted high order harmonics in HHG experiments along with the occurrence of hot electrons in certain interaction scenarios—have been obtained by combining deep neural networks (DNNs) and convolutional neural networks (CNNs) [53] with ensemble learning [54–56]. The DNNs and CNNs were built by grid search [57, 58], in conjunction with dropout [59–62] and constructive learning [63–67], with the CNNs exhibiting somewhat better performances in terms of speed and comparable accuracy in estimations. The chapter offers a comparative discussion of these alternate predictive modeling solutions, highlighting the performance improvement gained by deploying each combination of advanced machine learning and deep learning algorithms. Moreover, a significant part of this analysis is devoted to the challenges, advantages, caveats, accuracy, easiness of usage and suitability to the actual interaction scenario of these systems.

The last section proceeds to arguing the implications of big data and AI based predictive modeling for the scientific community, its potential, not only in joining together experimental observations, theory and simulation data, but also the potential and future prospects in deriving meaningful analysis and recommendations out of the already available information.

2. Big data and deep learning based predictive modeling for laser-plasma interaction

2.1. Opportunities and challenges for predictive modeling systems

The emergence of cloud computing and of open source big data designated platforms like Hadoop, Spark [68] and the framework ROOT [69, 70], along with the rise of deep learning [71–74] have rendered data processing and analysis trivially inexpensive. Massive amounts of a wide variety of information can today be interpreted at an unprecedented rate of speed. The consequence is particularly important for science because of various reasons. Firstly, migrating from expensive in-house computing systems to infrastructure as a service (IaaS) significantly cuts costs with capital investment. Secondly, the increased storage capacity and computer power make the cloud ideal for scientific big data applications development [75], specifically for statistics, analytics and recommender systems. Furthermore, workload optimization strategies can easily be incorporated in order to use the resources to maximum capacity. For applications that are both computational and data intensive the processing models combine different techniques like in-memory big data [76] or combined CPU—GPU processing.

Predictive modeling in a continuously evolving field like laser-plasma interaction is challenging from several points of view, mainly because this is an area previously unexplored with machine learning techniques and smart agents. Simulations serving this purpose have to this day relied

almost exclusively on codes that calculate according to various theories and approximations, hence on programmed software not on software that adapts and learns from experience and common knowledge. ROOT remains the only physics designated package that took some efforts in this new direction. Although it is mainly oriented towards signal treatment techniques and statistics, ROOT also incorporates machine learning algorithms to a lower extent.

Designing an intelligent predictive or recommender system for laser-plasma interaction should take into consideration quite many aspects. The start point and, at the same time, a central decisive factor in the design is actually the available interaction data, its amount and its structure. Specifically, interaction data for a particular kind of experiment is mostly heterogeneous in the sense that it can comprise experimental findings along with simulation yields and literature references, a situation bound to pose potential problems in terms of hardware, software environments and applicable machine learning paradigms. Storing and converting the available information in the same file format—especially if we are talking about terabytes or petabytes—is a time consuming operation. This caveat may be conveniently mitigated by using the NoSQL databases, a notable feature of big data platforms such as Hadoop or Spark. Furthermore, the NoSQL is schema-free, therefore facilitating structure modifications of data in applications. Through the management layer, data integration and validation can be easily attained. A second aspect of interaction data concerns features like inconsistency, incompleteness, redundancy or intrinsic noisiness. For a particular kind of experiment (e.g. a certain type of laser interacting with a specific target, in a predefined interaction configuration) there might be multiple results due to the fact that the same experiment was performed in different laboratories across the world. Consequently, the above mentioned data characteristics can be explained through the differences in diagnostic equipment or in its placement, through slight variations in the interaction configurations, in target compositions or the type of optical components. Simulations performed with different codes or theoretical estimations might also exist in the literature. Two other possible situations concern unavailable data and divergent or conflictual reports. Such variety entails various signal processing techniques like reduction, cleaning, filtering, integration, transforms and interpolations in order to remove noise, correct the inconsistencies and improve the decision-making process. However, these operations can be important consumers of resources, so they should be performed via distributed computing in conjunction with fast analytics purpose tools such as Apache Impala [77] and Apache Kudu [78].

Further applying machine learning algorithms [79] on this type of extended sets complicates things even more, firstly because we are talking about large volumes of data (at least 1 TB and easily up to several hundreds of TBs), and secondly because training even classical multilayer perceptrons (MLP) [80–83], self-organizing maps (SOM) [84, 85] and especially support vector machines (SVM) [86, 87] on conventional computers renders the process extremely difficult. Practically, this is a striking argument in favor of the custom-made clouds that provide not only computing power but also modularity, scalability and resilience. Beyond Hadoop's substantial parallelization, jobs dispatching and resource allocation capabilities, considerable speedup may be achieved within the Spark environment, owing to its graph technology. Built-in Mahout and MLib [88] machine learning libraries integrate a lot of the commonly deployed algorithms allowing the user to modify or add any new self-written modules. Within these frameworks, a common MLP can easily evolve towards deep learning due to the fact that multiple hidden layers (or cascaded MLPs) are no longer an impediment to fast training and

rapid convergence. The grid search algorithm permits testing multiple MLP topologies for the best performance on training and test sets, subsequently returning the best one. When stating multiple, an order of a few tens is perfectly feasible. Another useful tool, the dropout methods, randomly exclude various neurons along with their incoming and outgoing connections in order to achieve performance improvements and to avoid overfitting [89–91]. Some versions do not drop out units but just omit another portion of training data in each of the training cases, ultimately "averaging" over all the yielded MLPs (structures with the same topology but different weights, a consequence of the variation in the training set). This approach mitigates both, overfitting and potential falloffs or stagnations in the learning rate, effects associated primarily with sets featuring high percentages of redundant data. Other algorithms apply the "averaging" over networks with dropped out units or over many networks with different weights instead of merely considering the best configuration. Regardless of what is actually averaged, these solutions act similarly to ensemble learning and can be also combined with unsupervised techniques [92–94]. Inversely, constructive learning allows the user to add units or connections to the network during training, an approach known to be highly effective in escaping local minima of the objective function. All of the above classes of algorithms can be deployed for both CNNs and DNNs and, with slight modifications, even for 3D topologies of SOMs. Considerable boosts in terms of speed may be attainable through MapReduce acceleration [95] or GPU accelerated computing.

Practically, the choice of algorithms is of crucial importance when designing a predictive modeling system as they influence its overall performance both in terms of speed as well as in terms of accuracy and robustness. A high degree of modularity and scalability of the system is also desirable since it is fundamental to be able to add new algorithms and tools, or to replace others, as easily as possible without major reconfiguration and training issues. As new interaction data becomes available on a regular basis, retraining the system and its subsequent functionality are not supposed to be problematic. At the same time, hardware modifications within the cloud should only improve performances and not increase the risk of system crashes. Good predictions should be prevalent even when facing undesired events like hardware failures, software bugs and data corruption and from this point of view, the combination cloud-Hadoop-deep learning is ideal, mainly since Hadoop offers most of all resilience.

2.2. Engineering aspects of big data and deep learning based predictive systems

The development of intelligent systems with direct application in optimizing laser-plasma interaction experiments is a highly demanding task. Since it requires above all, enough hardware resources, the underlying infrastructure supporting the construction and deployment of the predictive systems was chosen to be a private cloud. Interaction with users is achieved via internet, hence, by extension one can consider this a "client–server" system, schematically displayed in **Figure 1**.

The "server-side" offers various functionalities in five areas. Firstly, it ensures the communication with users and handles the requests queues. Concerning the data management, the "server" is also responsible for the data storage, data manipulation and related processing operations. Thirdly, it stores and facilitates the incorporation of new software libraries. It provides computing power for establishing the optimal structure of the intelligent systems,

http://dx.doi.org/10.5772/intechopen.72844

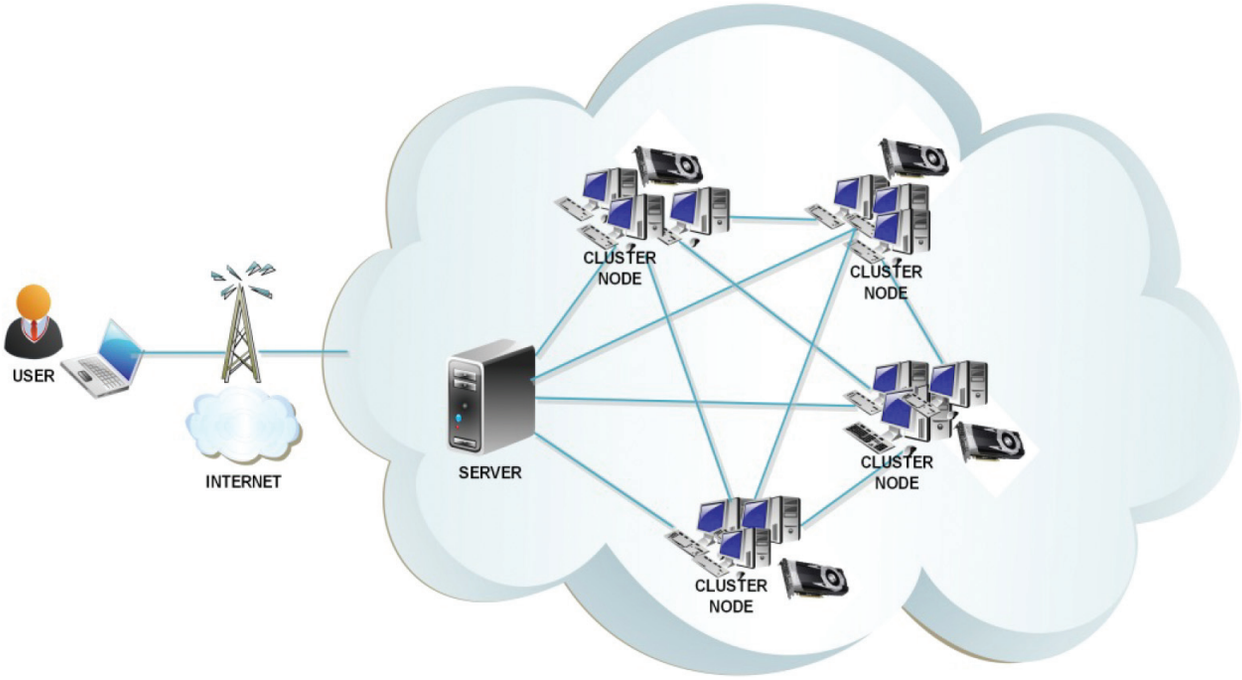

Figure 1. Client–server model of the supporting infrastructure. The server side is the private cloud on top of which resides Hadoop. It handles all tasks, from communications with the users to data storage, heavy computational tasks and ultimately, the deployment mode.

for training and validation. Last but not least, it supports the deployment mode of the validated predictive systems. At this stage, the users introduce the input parameters and obtain the predictions and/or recommendations. Among the advantages offered by a private cloud platform built using Hadoop are the rapid access to information, rapid processing and rapid transmission of results to the end user. But beyond processing and querying vast amounts of heterogeneous data over many nodes of commodity hardware, another significant advantage of the Hadoop streaming utility is the fact that it allows Java as well as non-Java programmed MapReduce jobs to be executed over the Hadoop cluster, in a reliable, fault-tolerant manner. The combination HDFS, HBase [96], Hive [97] and MapReduce is robust. Not only HDFS ensures data replication with redundancy across the cluster but every "map" and "reduce" job is independent of all other ongoing "maps" and "reduces" in the system. However, HDFS based data lakes lack what is a fundamental capability for complex applications that make use of the stored big data, and that is the random reads and writes capability. There is no point in trying to speed up data processing by developing new algorithms if accessing it translates into brute-force readings of an entire file system.

In this sense HBase was deployed on top of the HDFS data lake since it allows the fast random reads and writes that cannot be handled otherwise. As a NoSQL database, it is primarily useful because it can store data in any format. Additionally, HBase can also handle a variety of information that is growing exponentially, something which relational databases cannot. In other words, it supports the real-time updating and querying of the dataset which Hive does not and this is highly suitable for applying dropout and constructive learning on datasets. In contrast, Hive provides structured data warehousing facilities on top of the Hadoop cluster together with a SQL like interface that facilitates the creation of tables and subsequently, the storage of

structured data within these tables. Although, existing HBase structures can be mapped to Hive and operated on easily due to the efficient management of large datasets, inconveniently enough for certain cases, the data can be further used only in batch operations. The predictive modelings subject to this chapter use alternatively HBase and Hive as suitable to each of the combinations of algorithms. For a particular interaction scenario, the relevant information is extracted from the data lake, processed for cleaning and then stored into either HBase or Hive. As these sets of data are subject to MapReduce jobs and to machine learning, they may consequently suffer alterations, hence the modified versions are also written to the warehouse. Database dumps to HDFS are performed after each successful prediction experiment.

Within the cloud, the server is running Ubuntu Server 16.04 with MyEclipse 2015 Stable 2.0, Tomcat 8.5.5, JDK 8, release 1.8.0_102, Hadoop 2.7.3, HBase 2.7.3, Hive 2.0.0 installed. User requests are handled via JDBC (with Phoenix for HBase accessing) while the communication with the user is done via servlet developed in MyEclipse. Each of the four cluster nodes consists of six PCs, connected to a switch and each having a QuadCore CPU, a hard drive (1 TB, 6 Gbps, 7200 rpm, 32 MB cache), 16 GB of RAM and a 1000 Mbps full duplex connectivity card. Additionally, four GeForce GTX Titan with 2688 CUDA cores and 6 GB memory were attached to the cluster, one by node, their intended purpose being to facilitate the deployment of the deep learning algorithms. GPU computing is reputed for being well suited to the throughput-oriented workload problems that are characteristic to large-scale data processing. However, integrating GPUs within a Hadoop cluster is not obvious. While, parallel data processing can easily be handled by using several GPUs together or by GPU clustering [98], implementing MapReduce on GPUs has enough limitations [99] and requires a lot of finagling. For example GPUs communicate with difficulty over a network, hence being recommended to function with an Infiniband connection. Moreover GPUs cannot handle virtualization of resources. Their system architecture is therefore not entirely suitable for MapReduce without excessive modifications [98] and, up to recently, GPU and Hadoop were not even compatible. Therefore, to keep things as uncomplicated as possible, MapReduce tasks were entirely handled by the CPU nodes at all times.

After multiple machine learning experiments performed on earlier versions of this cloud [100, 101], observed performances have triggered—apart from hardware upgrades—several other tunings towards its overall optimization and in preparation for applying deep learning on the interaction data sets. These modifications address issues related to increasing the speed of processing raw data along with the speed of MapReduce tasks, decreasing the associated latencies by using fast analytics designated tools and an efficient management of workflows and finally, the containerization of tools and applications. In the design phase of a big data based complex application, special attention is to be given to the way jobs are planned and executed as this contributes to a large extent to the software's performances. For this purpose, workflow engines are a very useful tool as they schedule jobs in the data pipelines ensuring that they are ordered by dependencies. A workflow engine tracks each of the individual jobs and monitors the overall pipeline state. Built-in kill/suspend/restart/resume capabilities bring-in considerable improvements by helping diminish the potential bottlenecks caused by failed and downstreamed jobs. There are quite a few workflow engines available but for integration with Hadoop, the most stable and flexible are Oozie [102], Azkaban [103], Luigi [104], Airflow [105] and Kepler [106]. Criteria for choosing between these take into account the way

workflows are defined (configuration-based or code-based), the available support for various job types and its extensibility, the extent to which the state of a workflow may be tracked and most importantly, the manner in which the engine handles failures.

For the sake of simplicity and easiest integration, Oozie 4.2.0 was incorporated leading to a significant increase in the efficiency of all extract-transform-load (ETL) type of jobs as well as of the MapReduce ones. In spite of the lengthy and uneasy XML definition of workflows (configuration-based) and of individual jobs, Oozie is the only one that has built-in Hadoop actions, therefore enjoying the best compatibility with the Hadoop environment and the highest number of supported job types. Additionally, customized job support may be further integrated via available plugins. Within Oozie, workflow jobs are directed acyclic graphs (DAG) specifying a sequence of actions to be executed at certain time intervals, with a certain frequency and according to data availability. Recurrent and interdependent workflow jobs that form a data application pipeline are defined and executed through the Coordinator system. For a more efficient management, supplementary preventive or mitigating actions were coded in the coordinator application in order to cope with situations occurring due to partial, late, delayed or reprocessing of submitted data. A customized Java client that connects to the Oozie server was developed in order to monitor within the user interface, first of all, the workflow DAGs together with the corresponding states and secondly, to view and restart the failed tasks as soon as a notification in this sense is received. Since Oozie does not provide automatic notifications of failed jobs, this feature had to be implemented.

System resources are allocated to the jobs by YARN [107] with included optimizations in terms of efficiency and speed. YARN provides extensive support for long-running state-less batch jobs and analytical processing workloads such as machine learning algorithms. The containerization approach enhances even more these features however it does not rise to the same level of performance as Docker [108], and in this sense, it would be helpful to be able to install and deploy some other containerization technology on Hadoop in order to package applications and dependencies inside the container, to have a consistent environment for execution and, at the same time, enjoy the isolation from other applications or software installed on the host. The combination workflow engine—containerization is attractive for several reasons. First of all, it provides increased control both in the development phase as well as over the big data deployments. Secondly, it reduces significantly the rate of failed or stalling jobs and it offers uniformity and efficiency in resource allocation and resource sharing between different applications by orchestrating and organizing containers across any number of physical and virtual nodes. A containers' orchestrator mitigates the effects caused by failing nodes, adding more nodes or removing nodes from the cluster and by moving the containers from one node to another to keep them available at all times. Unfortunately, associating Hadoop with other container technologies than YARN is cumbersome as this system is not easily able to delegate the clustering functions to an external tool such as a container orchestrator. For instance, the particular installed version of Hadoop together with Docker for YARN grant the YARN NodeManager the possibility to launch YARN containers into Docker containers according to users' specification. However, this feature has certain caveats in terms of software compatibilities. Furthermore, the Docker Container Executor runs only in non-secure mode of HDFS and YARN and it requires Docker daemon to be running on the NodeManagers and the Docker client installed and able to start Docker containers. To prevent timeouts while starting jobs the

Docker images that are to be used by a job should already be found in the NodeManagers. Therefore, a reasonable compromise was met by installing the Docker Engine Utility only on the GPU nodes—without the YARN compatibility mode—with containers incorporating the deep learning libraries, including cuDNN.

Additionally, optimizations in terms of speed and latency mitigation within MapReduce tasks and the raw data processing and analysis are mainly due to Apache Tez [109] installed and configured atop of HDFS. Within a complex system such as a Hadoop cluster, latencies are common, inevitable and may have a variety of causes like storage I/O operations, network communications, architectural design imperfections or running software. Some latency is also inherent when launching jobs. As we have seen above, these latencies can be partially diminished by efficient resource allocation combined with scheduling of jobs. For MapReduce, its startup time is known to be one of the main sources of latencies, further performance enhancements being achievable by improving the dataflow processing and transmission from one stage to another. In this sense, the objective is to completely decouple the execution of the "mapper" from that of the "reducer" and have a direct output transmission from "mapper" to "reducer", with all "mappers" and "reducers" working in parallel. This approach might alleviate latency in jobs completion by up to 25 percent but unfortunately it tends to impact on the fault tolerance. Basically, a global sorting is potentially time-consuming—even when using multiple "mappers"—but it should be avoided mainly as this approach triggers by default the deployment of only one "reducer" which is very inefficient for large data sets.

An alternative strategy implies spilling files with intermediate results from "mapper" to "reducer" in order to preserve a certain degree of fault tolerance. Known as adaptive load moving, this technique leverages on a buffer attached to the output of each "mapper". On filled buffers, a combining function is applied for sorting purposes and the data is "spilled" out to storage. The spilled files are next adaptively pipelined to the "reducers" according to an "avoid overloading" policy and to a spilled files merging perspective. Fault tolerance is hence improved by reducing the risk of "mapper failure" which in turn limits the reducer's ability to merge files and process the information. Adaptive load moving applied to every "mapper" and "reducer" within the Hadoop cluster is better used in conjunction with process pooling for both the master and the worker nodes resulting in a significant spare of memory. Apache Tez was therefore employed to implement this strategy and to further improve other MapReduce related issues. For example, working with Hive and MapReduce often turns into costly operations and latencies of order of at least minutes, especially when executing a join, with "sky-high" query execution time and resource consumption. The data is often sharded and distributed across the network, thus performing a join requires matching tuples to be moved from one machine to another and consequently causing a lot of network I/O overhead. Tez is the one that gives Hive the possibility of running in real time, the query performance improvement being on average 50%. The major advantages of using Tez relate firstly to the adjustable number of "mappers" and "reducers" and secondly to the possibility of using the built-in cost-based query plan optimizations. Prior to executing a query, Tez determines the optimal numbers of "mappers" and "reducers" and automatically adjusts these numbers on the way based on the amount of processed bytes. Using the "Compute statistics" statement, the number of "mappers" and "reducers" can be monitored along with their speed in completing the corresponding tasks. Hence, should a bottleneck appear, its point of origin can be easily identified.

http://dx.doi.org/10.5772/intechopen.72844

The high volumes of data employed here trigger high query execution times. Tez implements query planning by building up multiple plans and choosing the best one out of the available computed versions. Query plan optimization is constructed in steps, starting from containerization and multi-tenancy provisioning, continuing with vectorization and ultimately with the cost-based planning, evaluation of plans and picking up the optimal one. Multi-tenancy permits the re-use of a container within a query by releasing all containers idling for more than 1 second. Vectorized query execution implies performing operations like scans, aggregations, filtering and joins in batches of 1024 rows at once instead of row by row. Finally, cost-based optimization of query execution plans significantly improves running times and the consumption of resources by evaluating the overall cost of every query as resulted from its associated plan. The evaluation reveals the viable types of operations, computes the cost of each combination and determines to which extent an increased degree of parallelism speeds up the execution time while lowering the amount of commissioned resources and making use of their reusability as much as possible.

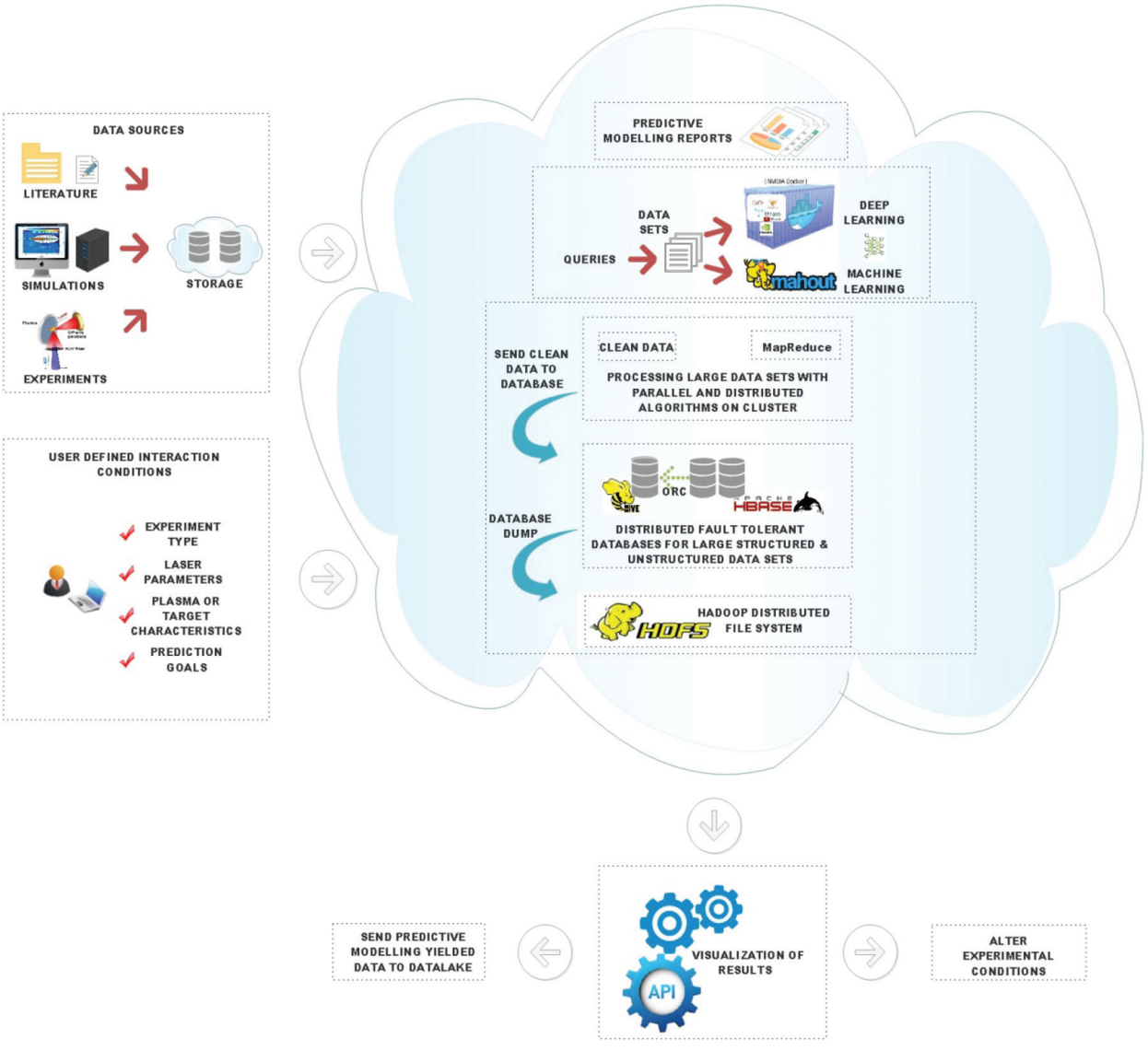

Figure 2. Conceptual design of an intelligent system that performs predictive modeling for laser-plasma interaction experiments.

Within a query, a MapReduce stage is followed by other stages. Tez checks the dependence between them and dispatches the independent ones to be executed in parallel. Another decision towards optimization concerns performing map joins instead of shuffle ones as the map joins minimize data movement and leverage on subsequent localized execution due to the fact that the hash map on every node is integrated into a global in-memory table and solely this table is being streamed, hence joins are made faster. A compromise has to be made, though, by provisioning larger Tez containers (much larger than the YARN ones) and by allocating one CPU and some GBs of memory per each of the containers. The performance of Hive queries can also be improved by enabling compression at the various stages, from table creation to intermediate data and final output. So, for these purposes a conversion to the ORC file format was done as these files result in 78% compression as compared to the initial text ones. Therefore, a search through 1 TB of data brings now only 5 seconds of latency.

Finally, to a reasonable extent, data intensive workloads also benefit from in-memory processing. Tez allows speculative executions to be attempted on faster nodes according to the Longest Approximate Time to End (LATE) strategy. These approaches were found to result in an overall speed performance improvement between one and one and a half orders of magnitude. In the case of iterative jobs, such as cost based function optimizations, an alleviation of up to 20 times in latency was obtained.

This subsection has so far been discussing just the underlying infrastructure used for building the predictive systems for laser-plasma interaction experiments optimizations, focusing not as much on the hardware but on the tools and tricks deployed for making the big data processing run faster and on less resources. However, some attention must be given also to the conceptual design of the predictive systems. This is displayed in **Figure 2**.

3. Migrating from machine learning algorithms to deep learning

3.1. First attempts in building predictive systems for HHG experiments

The particular cases of HHG experiments that were envisaged refer to the interaction of ultra-short and intense laser pulses with overdense plasmas (plasmas with density higher than the critical density). At the most basic level, this mechanism can be understood as the reflection of the incident laser and of its subsequently created harmonics on the oscillating plasma surface (oscillating mirror model OMM [110]). Since the plasma density is higher than the critical one, the laser cannot penetrate the plasma and thus it reflects on its surface. This surface is not flat and it exhibits an oscillatory movement due to the laser-induced heating mechanisms. While it is true that the yielded spectra depends a lot on the on the initial conditions—laser intensity, pulse duration, incidence angle, plasma density—the key factor is in fact the optimization of the resonance absorption as this fundamental process may account for up to 30% of the laser energy being absorbed by the plasma. Practically, the incident electromagnetic wave excites a plasma electron wave of the same frequency and the second harmonic results out of the mix between the plasma electron wave and the electromagnetic laser pump, hence its frequency being the double of the incident wave's. Although the second harmonic is mainly reflected, part of it can propagate inside the plasma and excite a wave of the same frequency, that in turn, by mixing with the

incident laser pump yields the third order harmonic. Moreover, it was also demonstrated that there is a correlation between the nonlinear, ponderomotively driven plasma surface motion and the production of energetic electrons [111, 112]. A pronounced asymmetry of longitudinal oscillations in a steep density profile is known to lead to wave breaking which in turn causes fractions of electrons to be irreversibly accelerated into the target. This kinetic process results in further absorption of energy from the laser. Furthermore, the accelerated fast electrons can themselves drive Langmuir waves, in the overdense region as well as in the ramps that form in front of the target, eventually leading to the generation of harmonics. This mechanism, namely, coherent wave excitation (CWE) [113] is the main responsible for HHG at moderate intensities. Further increase in laser intensities improves the prospects for efficient surface high order harmonics generation and, in principle, with relativistic lasers, high harmonics intensities may even exceed the intensity of the focused pulse by several orders of magnitude.

The goal of developing and deploying predictive modeling for HHG experiments was to have an estimate of the maximum order of the highest observable harmonic, along with the intensity, duration, wavelength of the various high harmonics and their conversion efficiency, given a particular laser interacting with a particular kind of plasma. The available data set consisted mainly of simulation data obtained by running various PIC codes but also from experimental data collected from the published scientific literature. Initially the data set amounted to 2 TBs but with the passing of time it reached about 5 TBs so the last predictions using deep learning were performed taking full advantage of the 5 TBs.

The first attempts in performing predictive modeling for high order harmonics generation experiments [100, 101] involved, on one hand commodity hardware with lower performances than the cloud currently used, without any GPUs and, on the other, an earlier version of Hadoop, installed and configured without any of the optimizations introduced in the meantime. This combination implied, first of all, long running times –up to several hours—just for MapReduce and further ones for the machine learning algorithms implemented with Mahout. Each additional TB of data was yet another challenge for the system and its available resources. Supervised learning made an obvious choice, consequently the most popular of the universal functional approximators [114], the MLP, was chosen as a starting point due to its versatility. Using its famous backpropagation algorithm (BKP) [115, 116] for error minimization during training, the MLP solves problems stochastically being able to provide approximate solutions even for extremely complex tasks. The high degree of connectivity between the nodes and the increased nonlinearity of this neural network cause its generalization ability to be among the best, coping rather well even with noisy and missing data. However this comes at the expense of significant running times in the training phase. While increasing the number of hidden layers is likely to lead to the improvement of overall performances, potentially revealing key features embedded in the data, adding too many of them was beyond the old system's capabilities, thus bottlenecks were reached very quickly.

The training set's input values are the laser intensity, laser wavelength, pulse duration, polarization, incidence angle and the type of plasma (introduced as ionization degree and elemental Z number) and its initial density. The desired output values in the training set are the maximum order of the highest observable harmonic, intensity values for different harmonics (including the highest one), harmonics' wavelengths, durations as well as their conversion

efficiencies. About 85% of the entire data formed the training set while the rest served as a test set and these percentages were hanged on to during the whole time up to the latest deep learning implementations. Multiple MLP topologies were tested, with different types and numbers of neurons, different numbers of hidden layers, batch or incremental training with various optimization algorithms. Deciding upon the number of neurons in the input layer depends mainly on the number of parameters that define a laser-plasma interaction scenario. The number of neurons in the output layer is generally a function of the yields that need to be classified or predicted. The number of hidden layers and the number of neurons within a layer were empirically determined. Hence, three of the investigated MLPs—henceforth labeled MLP1, MLP2 and MLP3, respectively—were found to exhibit satisfactory behavior in terms of accuracy. However the running hours were discouraging especially since, according to the results, it was obvious that an upgrade towards adding more hidden layers and more neural units was imminent. MLP1 has an input layer consisting of 8 Adaline neurons, two hidden layers, each with 12 sigmoidal neurons and an output layer of 5 sigmoidal units. It was trained with batch training, while the cost function was defined in terms of mean squared error (MSE) and optimized with Steepest Descent. MLP2 has three hidden layers, each with 10 sigmoidal neurons. The second difference from MLP1 is that its cost function was optimized with resilient backpropagation. Finally, MLP3 has two hidden layers, each with 11 sigmoidal units and it deploys the Levenberg-Marquardt algorithm for finding the global minimum of the cost function. For two HHG scenarios, **Table 1** displays the prediction results obtained with each of the three MLPs. Within the first scenario, laser's parameters are as follows: $I = 2 \cdot 10^{18}\ \mathrm{W/cm^2}$, $\lambda_0 = 800\ \mathrm{nm}$, polarization p, pulse duration $\tau_0 = 150\ \mathrm{fs}$, incidence angle $\alpha = 45^{\circ}$, interacting with an aluminum overdense plasma of electronic density equal to $n_e = 4n_c = 6.875 \cdot 10^{21}\mathrm{cm^{-3}}$. For the second scenario, the laser parameters are: $I = 10^{19}\mathrm{W/cm^2}$, $\lambda_0 = 800\ \mathrm{nm}$, polarization p, pulse duration $\tau_0 = 100\ \mathrm{fs}$, incidence angle with the plasma surface $\alpha = 60^{\circ}$, while the aluminum plasma has a density of $n_e = 8n_c = 1.375 \cdot 10^{22}\mathrm{cm^{-3}}$. The obtained predictions were in good agreement with PIC simulations as well as the literature data. However, it is easy to notice that the predicted intensities of the highest observable harmonic are lower in comparison to both theory and PIC results. This is caused by several factors, one of them being the heterogeneity of the available interaction data and the fact that the sets were minimally processed for cleaning during the "machine learning stages". As the collected information originates from multiple sources, it is obvious that the errors affecting the recorded values have different distribution functions. Furthermore, for a particular interaction scenario, we may have several experimentally determined values for the intensity of the highest observable harmonic and several numerical results. This constitutes redundant data, its principal negative effect being the overfitting. For the MLP based predictive modeling, all the redundant data was kept as it was, without any merging or advanced filtering. Overfitting is known to produce unrealistic predictions in MLPs even with noise free data, let alone with redundancy or sparsity. On the other hand, for certain scenarios, there was no available reference. Hence, the problem of missing information was solved by running a modified version of LPIC++ and recording the corresponding yields. In spite of having applied sampling and some filtering in order to assemble equilibrated training sets, a certain degree of incipient overfitting was detected in case of MLP1 and MLP2, thus some relative underestimation or overestimation was to be expected.

http://dx.doi.org/10.5772/intechopen.72844

	Highest observable harmonic				
	Max. Ord.	**Intensity W/cm²**	**Duration fs**	**Wavelength nm**	**Conv. efficiency**
Scenario 1					
Lit. Data	50	$2 \cdot 10^{11}$	20	16	10^{-7}
PIC Data	58	$2.1 \cdot 10^{11}$	19	13.8	10^{-7}
MLP1	54	10^{11}	21	14.4	10^{-7}
MLP2	56	10^{11}	20	14.8	10^{-7}
MLP3	52	10^{11}	19	15.4	10^{-7}
DNN1	52	$2.18 \cdot 10^{11}$	21	16.3	10^{-7}
DNN2	51	$2.1 \cdot 10^{11}$	19	15	10^{-7}
EL1	50.6	$2.02 \cdot 10^{11}$	20	15.8	10^{-7}
EL2	50	$1.98 \cdot 10^{11}$	20	16	10^{-7}
CNN1	51	$2.12 \cdot 10^{11}$	20	16.2	10^{-7}
CNN2	50.2	$2.04 \cdot 10^{11}$	20	15.4	10^{-7}
CNN3	50	$2 \cdot 10^{11}$	20	16.06	10^{-7}
EL4	50	$2.06 \cdot 10^{11}$	20	16	10^{-7}
EL5	50.4	$1.96 \cdot 10^{11}$	20	16.02	10^{-7}
Scenario 2					
Lit. Data	72	$2 \cdot 10^{11}$	12	11	10^{-7}
PIC Data	76	$2.1 \cdot 10^{11}$	11	10.5	10^{-7}
MLP1	74	$1.5 \cdot 10^{11}$	12	10.8	10^{-7}
MLP2	76	10^{11}	11	10.5	10^{-7}
MLP3	72	$2 \cdot 10^{11}$	12	11	10^{-7}
DNN1	74	$2.16 \cdot 10^{11}$	13	11.5	10^{-7}
DNN2	73	$2.08 \cdot 10^{11}$	11	10	10^{-7}
EL1	72.4	$2 \cdot 10^{11}$	12	10.8	10^{-7}
EL2	72	$2 \cdot 10^{11}$	12	11	10^{-7}
CNN1	73	$2.06 \cdot 10^{11}$	11	12	10^{-7}
CNN2	72.08	$2 \cdot 10^{11}$	12	11	10^{-7}
CNN3	72	$2 \cdot 10^{11}$	12	11	10^{-7}
EL4	72	$2 \cdot 10^{11}$	12	11	10^{-7}
EL5	72.05	$2.08 \cdot 10^{11}$	13	10.6	10^{-7}

Comparative results.

Table 1. Predictive modeling of HHG Scenarios 1 and 2.

Another aspect to be noted is the fact that all the MLPs discussed in this chapter feature hidden and output layers of sigmoidal units and this is the most important factor responsible for underestimation. The sigmoid activation function has a non-zero mean being prone to cause non-zero values in the Hessian matrix of the objective function, hence modifying the global minimum of the latter. A high number of sigmoidal neurons in a network strongly influences the weights adjustment during training, specifically, the corresponding weights in the last layers tend to take very small values (close to zero) and this saturation can last a very long time. To a good extent, the effect was mitigated by using a random initialization of weights, not only in the very beginning but also during the training process. Respectively, after observing a persistent saturation situation for a number of epochs, I performed some adjustments by adding small random values to the stagnating weights. This was found to improve the MLP's estimations on one hand and to increase the predicted values on another. Perhaps this was also one of the causes in the overestimation of certain parameters. A slightly better and more stable behavior was observed in case of MLP3, having required far less additive procedures of random values to the weights. Comparatively with the other two, the errors during training were smaller, the convergence faster and the predicted values for the high order harmonics were, in general, closer to the literature data, owing to the Levenberg–Marquardt algorithm, an algorithm known to improve the overall convergence speed due to the combination between Newton's method and Steepest Descent.

As stated above, on the course of interaction, the laser heats the plasma through various mechanisms. Inherently, some of the electrons acquire a lot of energy and become "hot", having very high temperatures, much higher than the plasma temperature. The percentage of hot electrons is very low but, in spite of this, their effects are not always negligible and, for certain experiments, even damaging. For an HHG experiment, a high percentage of hot electrons can disturb the oscillations of the plasma surface, a situation that affects the reflection of the laser, the CWE mechanism and consequently the HHG. For instance, a strong Brunel effect [111] leads to more thermal electrons. Consequently, it is important to have an accurate estimation of electron temperatures within the plasma along with the corresponding fractions of particles. For this purpose, another MLP (MLP4) was designed since the previous three gave only modest evaluations. Input values in the training set incorporate apart from the previously stated ones, the plasma's initial electronic temperature. The desired output values are electron temperatures accompanied by the estimated percentages of electrons that have these temperatures and the corresponding time moments. The best performing topology was found to be an MLP with 9 Adaline neurons in the input layer, 2 hidden layers, each with 11 sigmoidal units and an output layer with 3 neurons, also sigmoidal. The training was performed incrementally and the cost function defined in terms of MSE and optimized with Levenberg-Marquardt. For the same interaction conditions discussed above plus two additional cases (for Scenario 1, the incidence angle was modified to 30° from the normal to the plasma surface, this constituting Scenario 3 while for the same parameters in Scenario 2, the incidence angle was changed to 45°, this being labeled Scenario 4.) prediction results are shown in four graphs below. **Figures 3** and **4** display comparatively the percentage of electrons estimated to have a temperature above 10 keV at different time moments and above 100 keV, respectively. **Figure 3** refers to Scenarios 1 and 3, while **Figure 4** concerns the second and the fourth. The procedures of random initialization and adjustment (during training) of weights were also applied in an attempt of

http://dx.doi.org/10.5772/intechopen.72844

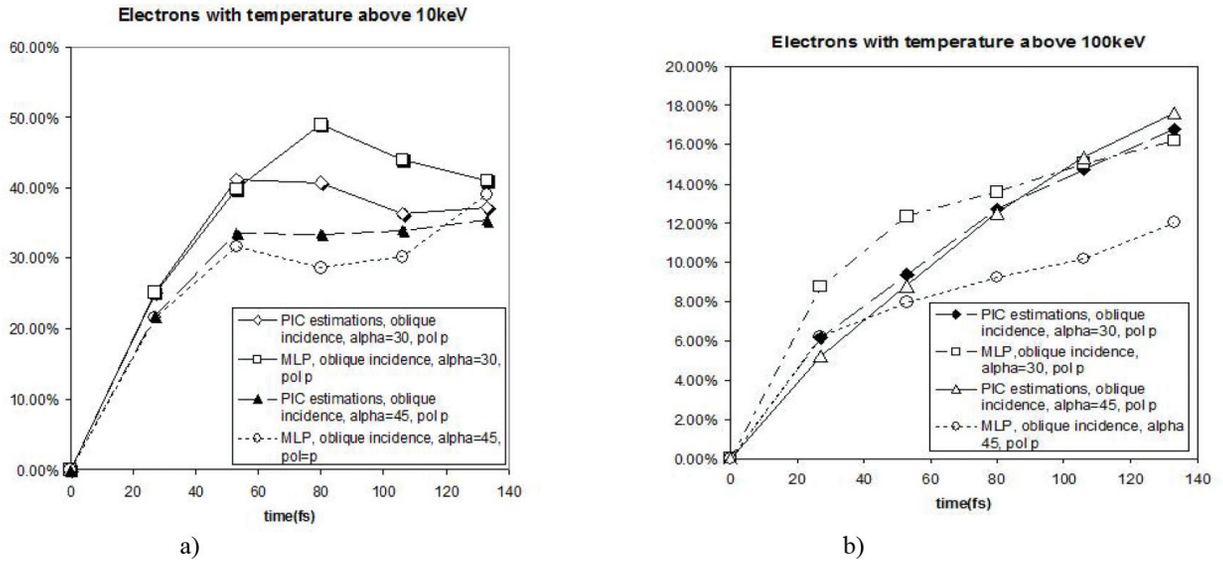

Figure 3. The variation in the percentage of electrons that exceed 10 and 100 keV, for interaction conditions consistent to Scenario 1 and Scenario 3. (a) Refers to the percentage of electrons that exceed 10 keV in temperature while. (b) Refers to those exceeding 100 keV.

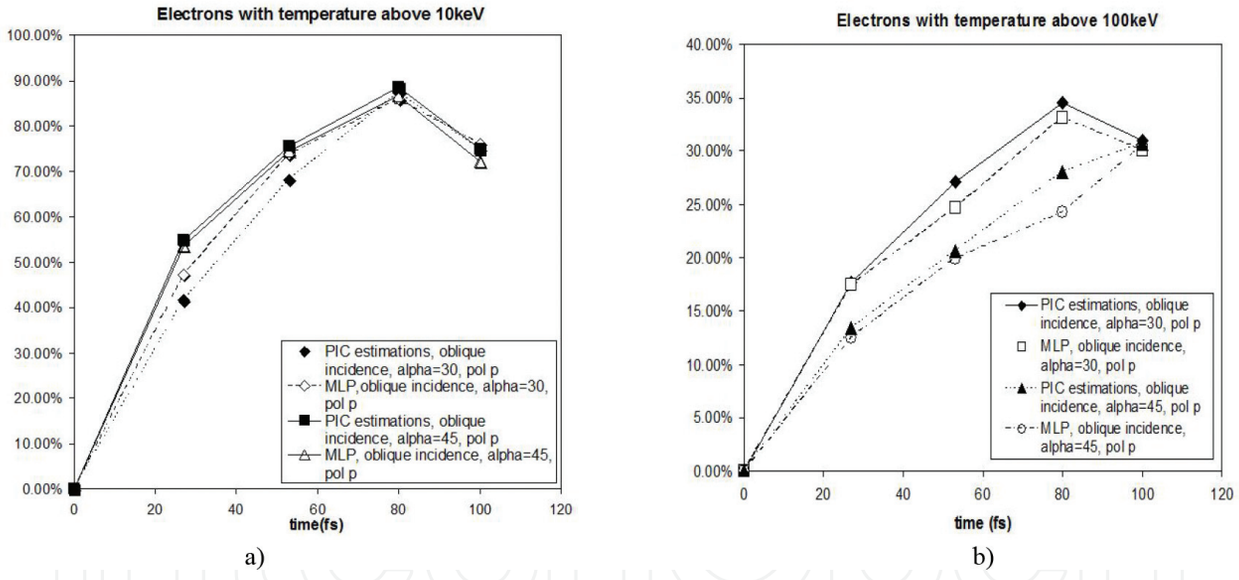

Figure 4. The variation in the percentage of electrons that exceed 10 and 100 keV, for interaction conditions consistent to Scenario 2 and Scenario 4. (a) Refers to the percentage of electrons that exceed 10 keV in temperature while. (b) Refers to those exceeding 100 keV.

improving MLP4's performance. However, it is the belief of this author that the combination between the network's topology, the sampling of available interaction data, the random additions and the incremental training, have led to some significant overestimations of the percentages of electrons (some 10%) in certain cases as values reported in the literature are smaller.

Prior to migrating towards deep learning, some trials were made with an unsupervised network, namely a SOM. The same training sets were used just that the data was differently organized, namely one entry in the training set consists of a 5×10 matrix. The matrix's columns stand for:

plasma (ionization degree, initial electronic temperature, initial plasma density, final plasma density, maximum plasma density), laser (intensity, wavelength, pulse duration, polarization, incidence angle) and 8 columns characterizing 8 different high order harmonics including the highest one (order, intensity, wavelength, duration, conversion efficiency). Several topologies were tested. However, just one of them yielded satisfactory results, namely a 2D network. The neurons' positions into the map were optimized based on Euclidian distance minimization and the competitive learning principle [117, 118]. SOM1 has a total of 16×21 nodes, disposed on a regular rectangular grid, with 16 nodes for mapping the harmonics' intensity and 21 for the orders of the harmonics. While a color code was employed for duration of pulses, the

Harmonic order	**Harmonic's characteristics**	**PIC (calculated)**	**MLP1**	**MLP2**	**MLP3**	**SOM1**
10	order	10	10	10	10	10
	intensity (W/cm^2)	$6 \cdot 10^{15}$	$4 \cdot 10^{15}$	$4 \cdot 10^{15}$	$6 \cdot 10^{15}$	$8 \cdot 10^{15}$
	duration (fs)	47	47	47	47	45
	wavelength (nm)	80	79.3	79.5	80.5	80
	conversion efficiency	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-3}
20	order	22	20	20	20	22
	intensity (W/cm^2)	$4 \cdot 10^{14}$	$3.5 \cdot 10^{14}$	$3.5 \cdot 10^{14}$	$4 \cdot 10^{14}$	$4 \cdot 10^{14}$
	duration (fs)	32	34	34	34	35
	wavelength (nm)	37	40	40	40	36.4
	conversion efficiency	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
30	order	34	30	32	30	28
	intensity (W/cm^2)	$5 \cdot 10^{13}$	$4.5 \cdot 10^{13}$	$5 \cdot 10^{13}$	$4 \cdot 10^{13}$	$4.1 \cdot 10^{13}$
	duration (fs)	26	27	27	27	28
	wavelength (nm)	23.5	26.7	25	26.7	28.6
	conversion efficiency	10^{-5}	10^{-5}	10^{-5}	10^{-5}	10^{-5}
40	order	46	42	44	40	38
	intensity (W/cm^2)	$4.5 \cdot 10^{12}$	$4 \cdot 10^{12}$	$4.2 \cdot 10^{12}$	$4.2 \cdot 10^{12}$	$6 \cdot 10^{12}$
	duration (fs)	22	23	23	24	25
	wavelength (nm)	17.4	19	18.2	20	21
	conversion efficiency	10^{-6}	10^{-6}	10^{-6}	10^{-6}	10^{-6}
50	order	58	54	56	52	49
	intensity (W/cm^2)	$2.1 \cdot 10^{11}$	10^{11}	10^{11}	10^{11}	$4 \cdot 10^{11}$
	duration (fs)	19	21	20	19	21
	wavelength (nm)	13.8	14.4	14.8	15.4	14.4
	conversion efficiency	10^{-7}	10^{-7}	10^{-7}	10^{-7}	10^{-7}

Comparative results for harmonics of orders 10, 20, 30, 40 and 50.

Table 2. Predictive modeling of HHG Scenario 1 using a SOM.

wavelengths and conversion efficiencies were derived computationally and written in an additional text file accompanying the map. The large number of nodes in this network is the consequence of the need for a better visualization of the final results. However, this weighs considerably in terms of number of training epochs and computation time and it was found that the SOM required far more resources than the MLPs and it took longer to train. In principle, it would be ideal to add more units and some improvements in terms of algorithms, along with the elimination of the accompanying text file and the associated computationally derived values. This basically means a SOM with more than two dimensions which, at the time, it was nearly impossible to implement. Hence, I desisted to pursue the development of predictive modeling using unsupervised learning. For exemplification, MLP performances in predicting high order harmonics and their features—for the interaction conditions described in Scenario 1—are displayed in **Table 2**, together with the SOM's and the results obtained from PIC simulations. The agreement between the forecasts of the MLPs and the ones of SOM is quite good, the values being within the same range.

3.2. Deep learning: Towards improved predictive systems for HHG experiments

In the view of building better predictive systems and even recommender systems for optimized laser-plasma interaction experiments, hardware upgrades were firstly made. Apart from adding an extra cluster node, replacing the storage hard drives with increased capacity ones in all computers and adding extra 8GB of RAM to all of them, a total of four GeForce GTX Titan were attached to the cluster, one by node. At the most basic level, deep learning networks can be viewed as modified MLPs that contain a high number of units and layers and are algorithmically more complex than the classical MLPs. Hence, the GPUs provide support for heavy computations. The Docker engine was installed on the GPU nodes along with the necessary Nvidia drivers and the nvidia-docker. A Docker image containing Theano, TensorFlow, Keras, Caffe, cuDNN and of course CUDA 8.0 and Ubuntu 14.04 was downloaded from GitHub, built and deployed as a container on the GPUs. All the deep learning based predictive modeling systems described in this chapter were discovered (structurally), trained, built and tested using these libraries. The optimal ones were implemented and deployed on the Hadoop cluster. The containerization of GPU applications provides important benefits such as reproducible builds, ease of deployment, isolation of individual devices running across heterogeneous driver/toolkit environments, requiring only Nvidia drivers to be installed. The images are agnostic of the Nvidia driver, with the required character devices and driver files being mounted when starting the container on the target machine.

The designation of the deep learning based predictive modeling systems were, for start, the same HHG experiments. However, the data lake increasingly incorporates other related interaction data. It is expected that more available information on what happens during various experiments performed in similar conditions will help to better understand the physics of interaction and consequently, to foresee what phenomena might occur. Huge data sets needed for training - after having been subject to MapReduce—have to be transferred to the GPU nodes. While the GPU memory system provides a higher bandwidth as compared to the CPU memory system, transferring data between the main memory and GPU memory is very slow. Copying via DMA to and from the GPU over the PCIe bus involves expensive context switches that reduce the available bandwidth considerably. This is why directives such as "gmp shared" and "gmp

private" have been added for identifying the data to be transferred between main memory and GPU memory. These directives are translated to relevant memory transfer calls, like cudaMalloc, cudaMemcpy, cudaFree within CUDA. Furthermore, potential redundant data transfers may slow down the GPU while running other jobs. These can be avoided through various dataflow and jobs workflow optimization techniques. For this reason, it was highly important to have the workflow engine and resource allocator configured and running on Hadoop. Additionally, the optimizations brought to MapReduce impact directly on the dataflow to GPUs.

The first deep learning networks that have been implemented were the DNNs. Since, basically, DNNs are MLPs with many hidden layers—commonly, a few tens—it was a relatively easy transition from machine learning to deep learning. In spite of this, things tend to get complicated when trying to guess out an optimal DNN configuration. This is a very tedious process. The solution comes from adopting a grid search algorithm combined with other two, namely constructive learning and dropout. This way, I was able to generate several hundreds of DNNs using constructive learning and dropout algorithms during the training phase and search for the optimal ones with grid search. Each of the tested configurations was cataloged and the best performance ones were prioritized for further usage. Both constructive learning and dropout can be performed in three ways, all of which have been tested. The first one involves adding more neurons to layers along with their corresponding connections to the others in the network (constructive learning) or simply removing ones (dropout) if performances are found to stagnate at an unsatisfactory level during the training phase. The training is continued and the evolution monitored. These actions, of adding and removing units may be performed several times during a training procedure. The second approach involves keeping the same network configuration while applying the algorithms on the data set instead of layers. Hence, instead of adding or removing units, one adds more data or removes portions of it from the training set. Last but not least, the third method is a combination of these, namely the construction and dropout procedures can be applied to both the network and the data. Although this is the most costly strategy, both in terms of resources as well as in terms of running times, it was by far the most effective one, yielding the best performances. This latter approach was also the one chosen for building the DNN based predictive systems.

Out of the huge pool of networks (nearly 500), two deep neural networks were found to perform better than all others. They will henceforth be labeled DNN1, respectively DNN2. DNN1 has an input layer consisting of 8 Adaline units, 20 hidden layers, containing only sigmoidal neurons. All layers have 12 units, except for the layers 3, 5, 6, 8 and 11. Layer 3 has 11 units, layer 5 has 15, layers 6 and 8 contain 12 each while layer 11 has just 7. The output layer features 5 sigmoidal neurons. DNN1 was trained with batch training and the cost function was optimized with Levenberg–Marquardt. DNN2 has an input layer consisting of 8 Adaline units, 36 hidden layers, containing only sigmoidal units. All layers are formed by 14 neurons, except for the layers 2, 6, 7, 9, 12, 16, 18, 23, 24, 25, 28, 30, 31, 32 and 35. Layer 2 has 15 units, layers 6, 9, 16, 25, 28 and 32 have 12, layers 7, 18 and 31 contain 13 each, layer 12 has 16, layer 23 has 15, layer 24 has 11, layer 30 contains 9 units while layer 35 has only 7. The output layer features 5 sigmoidal neurons. Training was performed also in batches and the cost function was optimized with Levenberg–Marquardt. For HHG Scenarios 1 and 2 discussed in the previous subsection, **Table 1** also includes the predictions obtained with DNN1 and DNN2. The following lines refer to predictions made with DNNs combined with ensemble

learning and these are labeled EL1 and EL2, respectively. EL1 and EL2 were obtained by applying ensemble learning on the best 50 configurations of all tested DNNs—this being the case of EL1—and, respectively over all configurations (EL2). This means that the predictions offered either by the 50 DNNs, either by all of them, were averaged arithmetically and the result used as the prediction value. Although it might not seem appropriate to use averaging, this algorithm has its foundations in statistics and it is expected to offer better performances than a plain DNN. Using ensemble learning also mitigates the underestimation problem caused by the sigmoidal neurons although this problem tends to be less pregnant in the case of deep neural networks due to their increased numbers of layers and units. Consequently the effect on the cost function optimization is not as strong. As a general conclusion, the predictions furnished by the DNNs and the DNNs combined with ensemble learning are much closer to the ones reported in the scientific literature than the values offered by the MLPs.

For Scenarios 2 and 4 presented in Section 3.1, the temperatures of the electrons within the plasma along with the corresponding percentages were predicted using DNN3 and EL3. **Figure 5a** displays the evolution of the electrons having temperatures above 10 keV, in terms of percentages, for Scenario 2 while **Figure 5b** refers to the same evolution but for conditions consistent with Scenario 4. **Figure 6a**, **b** present the variation of electron percentages for electrons having temperatures higher than 100 keV for Scenarios 2 (**Figure 6a**) and 4 (**Figure 6b**), respectively.

In each of the graphs four curves can be noticed. This is because the two curves corresponding to DNN3 and to EL3 are accompanied by the predictions of the MLP4 presented in the previous subsection and also by the results of PIC simulations. DNN3 has an input layer consisting of 9 Adaline units, 43 hidden layers, containing only sigmoidal neurons. All layers are formed by 15 neurons, except for the layers 4, 6, 9, 13, 15, 19, 21, 27, 34, 35, 38, 40 and 41. Layer 4 has 16 units, layers 6, 9, 19, 34, 35 and 40 have 12, layers 13, 15 and 27 contain 11 each, layer 21 has 17, layer 38 has 14, and finally, layer 41 has 11. The output layer features 7 sigmoidal neurons. The training was performed also in batches and the cost function was optimized with Levenberg–Marquardt.

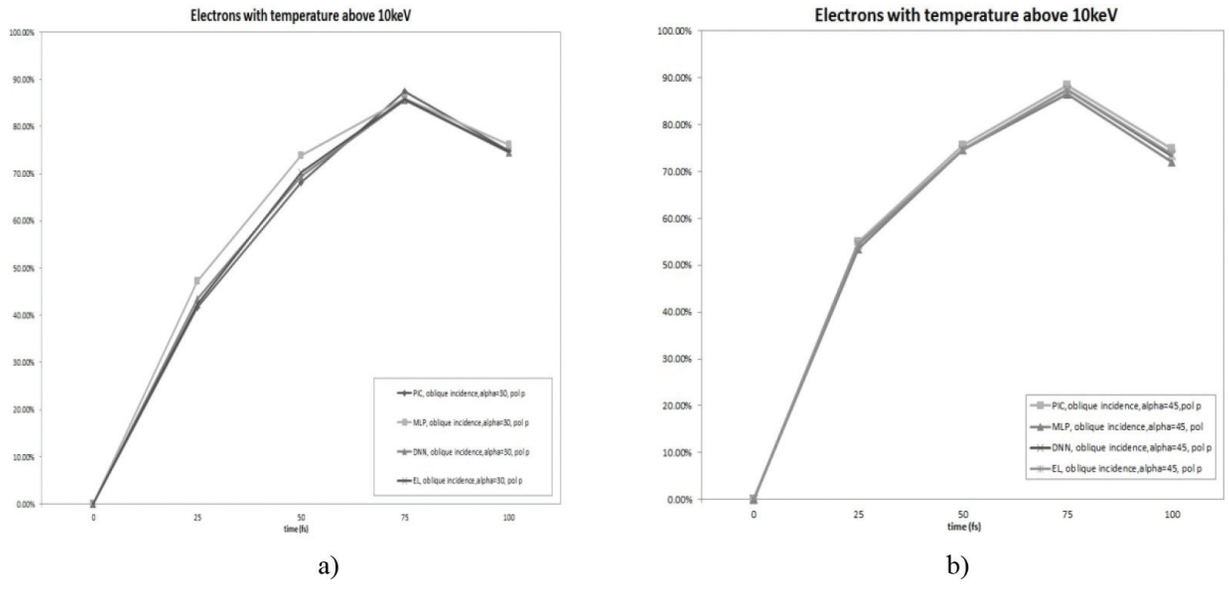

Figure 5. The variation in the percentage of electrons that exceed 10 keV, for interaction conditions consistent to Scenario 2 and Scenario 4. (a) Refers to Scenario 2 while (b) refers to Scenario 4.

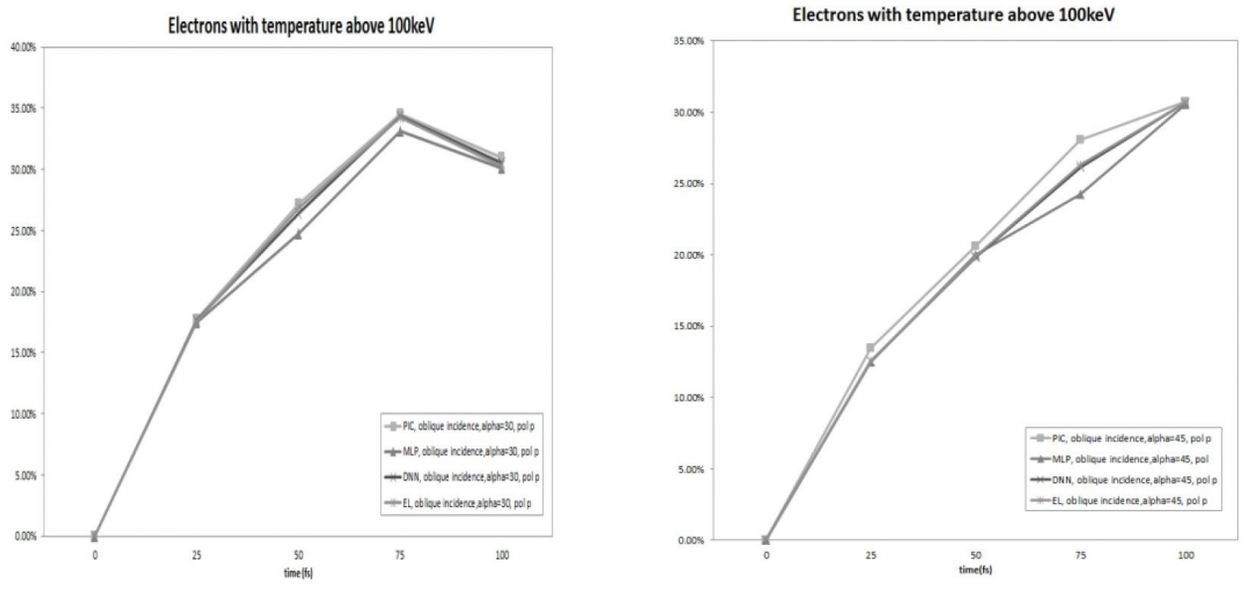

Figure 6. The variation in the percentage of electrons that exceed 100 keV, for interaction conditions consistent to Scenario 2 and Scenario 4. (a) Refers to Scenario 2 while (b) refers to Scenario 4.

EL3 was obtained by applying arithmetic averaging over a number of 100 predictions coming from the best 100 different DNN configurations that have been tested out of 478. Examining the curves, several conclusions can be drawn. Firstly, the DNN and the EL curves are very close, nearly superimposed. Secondly, the values predicted by DNN3 and EL3 are closer to the ones obtained from PIC simulations and more distanced from the predictions of the MLP. To the extent to which the PIC calculations are closer to real measurements, it can be confirmed that DNN and EL predictions are better than the MLP ones.

Since the obtained results were encouraging, further trials have been performed in the deep learning area, namely the deep neural networks were replaced with convolutional ones. CNNs are mostly reputed for their high suitability for applications dedicated to visual recognition from images. Therefore, in a way, CNNs' architectures make the explicit assumption that the inputs are images but this is not an incommoding aspect as—prior to being fed to a CNN—the values in the training and test data sets can be reorganized within an input volume formed out of laser parameters, plasma characteristics and yielded high order harmonics' characteristics just as images are normally structured. Consequently, I found a convenient way to organize the interaction information for the supervised training by making each entry in the training set a $20 \times 20 \times 20$ volume, in conjunction with a look-up table technique (LUT). The first dimension of each cube contains a reference in a LUT regarding the information on the incident laser's parameters, the second one includes references to the plasma characteristics (including electron and ion temperatures) while the last dimension has the references to high order harmonics spectra and to hot electrons' temperatures and percentages. The very nature of the CNN facilitates the incorporation of more features within the training and test sets. What distinguishes CNNs from DNNs is the fact that all of its layers have neurons arranged in three dimensions: width, height, depth. A second major difference concerns the connectivity. Within a DNN, all units are connected to all other neurons in the previous as well as in the next layer. As the number

http://dx.doi.org/10.5772/intechopen.72844

of layers rises, the number of connections grows exponentially, thus impacting dramatically on the computational resources. The CNNs bring a major change. The neurons in a layer are only connected to a small region in the layer before it. The output layer is the smallest in dimensions, as inherently, by the end of the network, the full input is reduced to a single vector of class scores arranged along the depth dimension. Three main types of layers exist within the architecture: convolutional layer, pooling layer and the fully connected layer and these are stacked together to form a CNN. The input is fed firstly to one or more subsequent convolutional layers. This layer is the core building block of the network and it performs all the heavy computations. More specifically, it calculates the output of neurons that are connected to local regions in the input, each of the neurons computing a dot product between its weights and a small region it is connected to in the input volume. The convolutional layer has as parameters a set of learnable filters, defined by the user. Every filter is small spatially (along the width and height dimensions), but extends through the full depth of the input volume (what in this particular case is the high orders harmonics spectra). Moreover, each of the filters is looking for a different thing in the input. During the forward pass, each filter is slid (convolved) across the width and height of the input volume and dot products between the entries of the filter and the input at any position are hence calculated. As the filter is slid, a bi-dimensional activation map is produced, that gives the responses of that filter at every spatial position. These activation maps are stacked along the depth dimension and produce the output volume which is next fed either to a pooling layer, either to a second convolutional layer. Intuitively, the network will learn filters that activate when they see some type of feature such as an increased number of high order harmonics or very intense ones on the first layer, or, eventually, an entire rich spectra on the higher layers of the network. The pooling layers perform a downsampling operation along the spatial dimensions (width, height), resulting in smaller volumes. Most commonly, they are periodically inserted in-between successive convolutional layers as they progressively reduce the spatial size of the representation in order to lower the amount of parameters and ease up the computational load in the network. But more importantly, pooling layers mitigate overfitting. The pooling layer operates independently on every input slice, most of the time by using the "max" operation. In addition to max pooling, average pooling or L2-norm pooling may be encountered. Historically, average pooling used to be the most popular but recently it has been progressively replaced by the max pooling as the latter was demonstrated to work better in practice. The fully-connected layer computes the class scores and packs them in a vector, each class score representing a high order harmonic with particular features. This is the only layer within which neurons are connected just as in a DNN. Their activations can hence be computed with a matrix multiplication followed by a bias offset. Basically, both the fully connected layer and the convolutional layer perform the convolution but the neurons in the convolutional layer are connected only to a local region in the input, and many of them share parameters in order to save computational resources.

As with the previous case of the DNNs, about 600 different CNNs have been generated and searched through with the aid of the grid search algorithm. To generate the configurations, several operations have been applied. Firstly, the number of convolutional and pooling layers was varied, as well as their position. For example, I constructed networks containing a pooling layer after each convolutional layer or a pooling layer after each two or three convolutional layers. In some network versions, pooling layers were absent except for a single one, just before the fully connected layer. Secondly, within each convolutional layer, the number of

filters was modified in order to observe what happens if the layer is sensitive to more features or if it is sensitive to features that are not relevant for all the types of HHG experiments. Thirdly, several pooling methods have been tested for the pooling layers in each network, namely, the classical max pooling, the average pooling and the stochastic pooling. Last but not least, the dropout and constructive learning algorithms were applied on the fully connected layer, resulting in more CNN configurations. For efficiency purposes, regularization methods such as L2 [119] and elastic net regularization [120] were applied to all the convolutional layers and to the fully connected layer when some of the weights were observed to peak excessively. The objective was to force the layers of the CNN to make use of all of their inputs at the same rate (as much as possible) rather than to use portions of their inputs preferentially. However, the risk is ending up in having a network layer with neuron weights that are "diffuse" and rather small. Elastic net regularization—a combination between L1 and L2 types—proved to be more efficient than either of the two. Ensemble learning was also deployed, just as before, averaging over either the predictions offered by all networks, either by applying the average on the best performing 10% of the configurations. The best performing three configurations are labeled CNN1, CNN2 and CNN3, respectively. All the networks take the same input size, namely the $20 \times 20 \times 20$ volume described above and were exposed to the elastic net regularization. Their configurations are as follows. CNN1 has four convolutional layers. The first one has 128 filters and a filter size of $5 \times 5 \times 20$, the second and third convolutional layers have 256 filters but a smaller filter size, more precisely $3 \times 3 \times 20$. Finally, the fourth convolutional layer has 512 filters and the same filter size as the latter two. After the first and the third layers, a pooling layer was introduced. The pooling layers use stochastic pooling. The network's architecture ends with a fully connected 3D cubic layer with 1024 units. It can be noticed that when applying a cubic root to this value, the resulting number of units on each dimension is not an integer. This is because, dropout and constructive learning were applied to the fully connected layer resulting in either vacancies, either insertion of neurons into the volume and in an overall addition of 24 units. The training of CNN1 was done in batches of 512 examples per gradient step with stochastic gradient descent used for the cost function optimization along with the bespoke backpropagation of errors. CNN2 has five convolutional layers, also optimized with elastic net regularization, the first four ones being identical to CNN1's. The fifth layer has, 512 filters, a filter size of $3 \times 3 \times 20$ and it is followed by the sole pooling layer of CNN2. This pooling layer also employs stochastic pooling. The network's architecture ends with two fully connected 3D cubic layers with 1024 units each but with different configurations of neurons within the layers' volumes. This is again due to dropout and constructive learning applied to the fully connected layers. The training of CNN2 was done in the same way but the cost function optimization was achieved via Levenberg–Marquardt. Last but not least, CNN3 has also five convolutional layers (elastic net regularization was applied to the weights), with the first layer having 126 filters and the same $5 \times 5 \times 20$ filter size. The second and the third layers have 252 filters, the second having a $5 \times 5 \times 20$ filter size and the third a $3 \times 3 \times 20$. The fourth and the fifth have 504 filters with the same filter size as the previous one. CNN3 has just one pooling layer in between the fourth and the fifth layers, which makes use of max pooling. The last convolutional layer is followed by two fully connected 768 units layers that were subject to dropout and constructive learning. The training was done also in batches, the stochastic gradient descent being employed with the AdaDelta adaptive learning method [121]. CNN1 and CNN2 use a stride of one for all the

convolutional layers while CNN3 uses a stride of 2 for the first and the fourth convolutional layers. This is a consequence of compromising based on the memory constraints that, at some point, bottleneck the GPUs. EL4 and EL5 are ensemble learning yields. EL4 averages over the best performing 10% of the CNNs while EL5 averages over all. For HHG Scenarios 1 and 2 discussed in the previous subsection, the last rows of **Table 1** feature the predictions obtained with the CNNs, EL4 and EL5.

For predicting the temperatures of the electrons within the plasma along with the corresponding percentages, it was found that the performances of CNN1, CNN2, CNN3, EL4 and EL5 were roughly identical and very close to those of DNN3 and EL3. In terms of running times, the convolutional neural networks take less time to train than the deep networks, the order being 50 hours less, on average. Prior to applying ensemble learning, the GPU Inference Engine (GIE) was used in the test phase to optimize the trained networks for run-time performance. Layer optimizations are attainable through GIE to the extent to which layers with unused output are eliminated in order to save computation time or layers may be fused for better overall performance.

One last comment concerns the libraries Theano, TensorFlow, Keras and Caffe. All four of these libraries have been alternatively used to implement both the DNNs and the CNNs. Running code written in TensorFlow was found to have the lowest running times, followed by those written in Caffe, Theano and Keras which took the most time to complete (taking 23 more minutes to complete). However, the differences are not that disturbing so perhaps this is due to some less optimal code. In terms of user friendliness, I found that the easiest to work with was Theano, followed by Caffe, TensorFlow and Keras. Again, this ranking is subjective since Theano was the first library I started working with. There is still a lot of work to be done and more room for improvements especially towards building recommender systems, hence prescriptive analytics, by combining CNNs with reinforcement learning policies. This would be of particular interest since such a system would issue a precise recommendation on how to adjust the interaction conditions in order to optimize a particular laser-plasma interaction experiment.

4. Conclusion

Technological advances in the field of laser -plasma interaction and diagnostics have provided the scientific community with lots of data. Within the last few years we have been experiencing a continuously upraising accessibility, not only to storage space and increased computer power, but also to a multitude of readily-built and easily modifiable open-source software libraries. It is thus becoming less and less problematic to exploit and explore this already available information in ways that have never been attempted before.

This paper proposes an alternative to the classical plasma kinetics simulations. Acknowledging the potential innovative technologies like cloud computing, big data, machine learning and, ultimately, the deep learning have for science, the author showed how these can be used for predictive modeling of laser-plasma interaction scenarios, with a focus on high harmonics generation. The deployment of the presented systems has the potential of yielding better predictive analytics and hence optimized laser-plasma interaction experiments, by offering a fair

estimation of interaction conditions or insights on different phenomena occurring during the laser-plasma interaction.

Acknowledgements

The author would like to acknowledge support from the National Authority for Scientific Research and Innovation under Program NUCLEU, project PN1647 LAPLAS IV.

Author details

Andreea Mihailescu

Address all correspondence to: andreea.mihailescu@inflpr.ro

Lasers Department, National Institute for Lasers, Plasma and Radiation Physics, Bucharest-Magurele, Romania

References

[1] Yanovsky V et al. Ultra-high intensity—300-TW laser at 0.1 Hz repetition rate. Optics Express. 2008;**16**:2109-2114

[2] Texas Petawatt Laser [Internet]. 2015. Available from: texaspetawatt.ph.utexas.edu/overview.php

[3] Vulcan Laser Facility [Internet]. 2015. Available from: www.clf.stfc.ac.uk/CLF/Facilities/Vulcan/12248.aspx

[4] Astra Gemini Facility [Internet]. 2015. Available from: www.clf.stfc.ac.uk/CLF/Facilities/Astra/12254.aspx

[5] PHELIX Laser Facility [Internet]. 2015. Available from: https://www.gsi.de/en/start/research/forschungsgebiete_und_experimente/appa_pni_gesundheit/plasma_physics phelix/phelix.htm

[6] Apollon Laser [Internet]. 2015. Available from: http://www.apollon-laser.fr/

[7] Kneip S et al. Bright spatially coherent synchrotron X-rays from a table-top source. Nature Physics. 2010;**6**:980

[8] McKinnie I, Kapteyn H. High-harmonic generation: Ultrafast lasers yield x-rays. Nature Photonics. 2010;**4**:149

[9] PIC Codes and Methodology [Internet]. 2015. Available from: http://plasmasim.physics.ucla.edu/codes

[10] Pfund RE et al. LPIC++ a parallel one-dimensional relativistic electromagnetic particle-in-cell code for simulating laser-plasma interaction. AIP Conference Proceedings. 1998; **426**:141

[11] Lichters R et al. Short-pulse laser harmonics from oscillating plasma surfaces driven at relativistic intensity. Physics of Plasmas. 1996;**3**:3425

[12] Verboncoeur JP et al. An object-oriented electromagnetic PIC code. Computer Physics Communications. 1995;**87**:199

[13] Burau H et al. PIConGPU: A fully relativistic particle-in-cell code for a GPU cluster. IEEE Transactions on Plasma Science. 2010;**38**(10):2831

[14] Brady C et al. EPOCH, an open source PIC code for high energy density physics, user manual for the EPOCH PIC codes version 4.3.4, University of Warwick, collaborative computational project in plasma. Physics. 2015

[15] Vsim [Internet]. 2016. Available from: https://www.txcorp.com/vsim

[16] Fonseca RA et al. OSIRIS: A three-dimensional, fully relativistic particle in cell code for modeling plasma based accelerators. In: Computational Science-ICCS 2002, Series Lecture Notes in Computer Science. Vol. 2331. Berlin/Heidelberg: Springer; 2002. pp. 342-351

[17] Fonseca RA et al. One-to-one direct modeling of experiments and astrophysical scenarios: Pushing the envelope on kinetic plasma simulations. Plasma Physics and Controlled Fusion. 2008;**50**:124034

[18] Fiuza F et al. Efficient modeling of laser–plasma interactions in high energy density scenarios. Plasma Physics and Controlled Fusion. 2011;**53**:074004

[19] Huang C et al. Quickpic: A highly efficient particle-in-cell code for modeling wakefield acceleration in plasmas. Journal of Computational Physics. 2006;**217**:658

[20] An W et al. An improved iteration loop for the three dimensional quasi-static particle-in-cell algorithm: Quickpic. Journal of Computational Physics. 2013;**250**:165

[21] Tzoufras M et al. A Vlasov-Fokker-Planck code for high energy density physics. Journal of Computational Physics. 2011;**230**:6475

[22] Tzoufras M et al. A multi-dimensional Vlasov-Fokker-Planck code for arbitrarily anisotropic high-energy-density plasmas. Physics of Plasmas. 2013;**20**:056303

[23] Owens JD et al. A survey of general-purpose computation on graphics hardware. Computer Graphics Forum. 2007;**26**:80-113

[24] Owens JD et al. GPU computing, graphics processing units-powerful, programmable and highly parallel—are increasingly targeting general-purpose computing applications. Proceedings of the IEEE. 2008;**96**:879

[25] Fatahalian K, Houston M. A closer look at GPUs. Communications of the ACM. 2008;**51**(10): 50

[26] GPU Applications: Hundreds of Applications Accelerated [Internet]. 2017. Available from: http://www.nvidia.com/object/gpu-applications.html

[27] Tesla GPU Accelerators for Servers [Internet]. 2017. Available from: http://www.nvidia.com/object/tesla-servers.html

[28] Decyk VK, Singh TV. Particle-in-cell algorithms for emerging computer architectures. Computer Physics Communications. 2014;**185**:708

[29] Suzuki J et al. Acceleration of PIC simulation with GPU. Plasma and Fusion Research. 2011;**6**:2401075

[30] Lu Q, Amudson J, Synergia CUDA. GPU-accelerated accelerator modeling package. Journal of Physics Conference Series. 2014;**513**:052021

[31] Decyk VK. Skeleton particle-in-cell codes on emerging computer architectures. Computing in Science & Engineering. 2015;**17**:47

[32] Abreu P et al. PIC codes in new processors: A full relativistic PIC code in CUDA-enabled hardware with direct visualization. IEEE Transactions on Plasma Science. 2011;**39**:675

[33] Decyk VK, Singh TV. Adaptable particle-in-cell algorithms for graphical processing units. Computer Physics Communications. 2011;**182**:641

[34] Germaschewski K et al. The plasma simulation code: A modern particle-in-cell code with patch-based load balancing. Journal of Computational Physics. 2016;**318**:305

[35] Abreu P et al. Streaming the Boris pusher: A CUDA implementation. AIP Conference Proceedings. 2009;**1086**:328

[36] Yang C et al. Fast weighing method for plasma PIC simulation on GPU-accelerated heterogeneous systems. Journal of Central South University of Technology. 2013;**20**:1527

[37] Stantchev G et al. Fast parallel particle-to-grid interpolation for plasma PIC simulations on the GPU. Journal of Parallel and Distributed Computing. 2008;**68**:1339

[38] Rossinelli D et al. Mesh-particle interpolations on graphics processing units and multicore central processing units. Philosophical Transactions of the Royal Society A. 2011;**369**:2164

[39] Wang P et al. A parallel current deposition method for PIC simulation on GPU. In: Proceedings of IEEE International Vacuum Electronics Conference (IVEC2015), IEEE, IEEE XPlore Digital Library; 2015. p. 7224036

[40] Kong X et al. Particle-in-cell simulations with charge—conserving current deposition on graphic processing units. Journal of Computational Physics. 2011;**230**:1676

[41] Rossi F et al. Towards robust algorithms for current deposition and dynamic load-balancing in a GPU particle-in-cell code. AIP Conference Proceedings. 2012;**1507**:184

[42] The ALaDyn PIC Suite [Internet]. 2015. Available from: http://www.physycom.unibo.it/aladyn_pic/

[43] FBPIC (Fourier-Bessel Particle-in-Cell Code) [Internet]. 2016. Available from: https://fbpic.github.io/index.html

[44] Kirchen M, Lehe R. Accelerating a Spectral Algorithm for Plasma Physics with Python/Numba on GPU, talk given at GPU Technology Conference GTC 2016. p. IDS6353

[45] Apache Hadoop [Internet]. 2017. Available from: https://hadoop.apache.org

[46] MapReduce Tutorial. Apache Hadoop 2.7.4 [Internet]. 2017. Available from: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[47] Apache Mahout. An environment for quickly creating scalable performant machine learning applications [Internet]. 2017. Available from: https://mahout.apache.org

[48] Lyubimov D, Palumbo A. Apache Mahout: Beyond MapReduce. Distributed Algorithm Design; 2016. ISBN-13: 978-1523775781

[49] Theano [Internet]. 2017. Available from: http://deeplearning.net/software/theano/

[50] TensorFlow [Internet]. 2017. Available from: https://www.tensorflow.org/

[51] Keras [Internet]. 2017. Available from: https://keras.io/

[52] Caffe [Internet]. 2017. Available from: http://caffe.berkeleyvision.org/

[53] Krizhevsky A et al. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems. 2012;**1**:1097-1105

[54] Opitz D, Maclin R. Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research. 1999;**11**:169-198

[55] Polikar R. Ensemble based systems in decision making. IEEE Circuits and Systems Magazine. 2006;**6**(3):21-45

[56] Rokach L. Ensemble-based classifiers. Artificial Intelligence Review. 2010;**33**(1–2):1-39

[57] Bergstra J, Bengio Y. Random search for hyper-parameter optimization. The Journal of Machine Learning Research. 2012;**13**:281

[58] Bao Y, Liu Z. A fast grid search method in support vector regression forecasting time series, intelligent data engineering and automated learning-IDEAl 4224 of the series. Lecture Notes in Computer Science. 2006;**4224**:504-511

[59] Srivastava N et al. Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. 2014;**15**(1):1929-1958

[60] Hinton G et al. Improving neural networks by preventing co-adaptation of feature detectors [Internet]. 2012. Computing Research Repository (CoRR) abs/1207.0580. Available from: https://arxiv.org/abs/1207.0580

[61] Baldi P, Sandowski P. The dropout learning algorithm. Artificial Intelligence. 2014;**210**:78-122

[62] Baldi P, Sandowski P. Understanding dropout. In: Proceedings of Advances in Neural Information Processing Systems (NIPS 2013). Neural Information Processing Systems Foundation, Inc; 2013. p. 4878

[63] Grochowski M et al. constructive neural network algorithms that solve highly non-separable problems. In: Franco L et al. editors. Constructive Neural Networks. Berlin: Springer-Verlag; 2009. pp. 49-70

[64] Campbell C. Constructive learning techniques for designing neural networks systems. In: Leondes CT, editor. Neural Network Systems, Techniques and Applications. San Diego: Academic Press; 1997. pp. 1-54

[65] Fahlman SE, Lebiere C. In: Touretzky DS, editor. The Cascade-Correlation Learning Architecture, Advances in Neural Information Processing Systems. Los Altos, CA: Morgan Kaufmann Publishers; 1990. pp. 524-532

[66] Littmann E, Ritter H. Learning and generalization in cascade network architectures. Neural Computation. 1996;**8**:1521-1539

[67] Kwok TY, Yeung DY. Constructive algorithms for structure learning in feedforward neural networks for regression problems. IEEE Transactions on Neural Networks. 1997;**8**(3):630-645

[68] Apache Spark [Internet]. 2015. Available from: https://spark.apache.org

[69] ROOT Data Analysis Framework [Internet]. 2015. Available from: https://root.cern.ch

[70] Computing at CERN [Internet]. 2015. Available from: http://home.cern/about/computing

[71] LeCun Y et al. Deep learning. Nature. 2015;**521**:436

[72] Bengio Y. Learning deep architectures for AI. Foundations and Trends in Machine Learning. 2009;**2**(1):1-127

[73] Goodfellow I et al. Deep learning. In: Dietterich T, Bishop C, Heckerman D, Jordan M, Kearns M, editors. Adaptive Computation and Machine Learning Series. Cambridge, MA: MIT Press; 2016

[74] Schimdhuber J. Deep leaning in neural networks: An overview. Neural Networks. 2015;**61**:85

[75] Argonne Leadership Computing Facility: Project Magellan: Cloud Computing for Science [Internet]. 2016. Available from: http://www.alcf.anl.gov/magellan

[76] Zhang H et al. In-memory big data management and processing: A survey. IEEE Transactions on Knowledge and Data Engineering. 2015;**27**(7):1920-1948

[77] Apache Impala [Internet]. 2017. Available from: https://impala.apache.org/

[78] Apache Kudu [Internet]. 2017. Available from: https://kudu.apache.org/

[79] Mohri M et al. Foundations of Machine Learning. Cambridge, MA: MIT Press; 2012

[80] Bishop CM. Neural Networks for Pattern Recognition. 3rd ed. Oxford: Oxford University Press; 1995

[81] Fine TL. Feedforward Neural Network Methodology. 3rd ed. NewYork: Springer-Verlag; 1999

[82] Haykin S. Neural Networks: A Comprehensive Foundation. 2nd ed. New York: Macmillan College Publishing; 1998

[83] Bishop CM. Pattern Recognition and Machine Learning. New York: Springer-Verlag; 2006

[84] Kohonen T. Self-organized formation of topologically correct feature maps. Biological Cybernetics. 1982;**43**(1):59

[85] Kangas JA et al. Variants of self-organizing maps. IEEE Transactions on Neural Networks. 1999;**1**(1):93-99

[86] Cortes C, Vapnik V. Support-vector networks. Machine Learning. 1995;**20**(3):273

[87] Ben-Hur A et al. Support vector clustering. Journal of Machine Learning Research. 2001;**2**:125-137

[88] Apache Spark MLib: Scalable machine learning library [Internet]. 2016. Available from: https://spark.apache.org/mlib

[89] Geman S et al. Neural networks and the bias/variance dilemma. Neural Computation. 1992;**4**:1

[90] Sarle WS. Stopped training and other remedies for overfitting. In: Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics. VA, Fairfax: Interface Foundation of North America; 1995. pp. 352-360

[91] Weigend A. On overfitting and the effective number of hidden units. In: Mozer MC, Smolensky P, Touretzky DS, Elman JL, Weigend AS, editors. Proceedings of the 1993 Connectionist Models Summer School. Hillsdale, NJ: Erlbaum Associates; 1994. pp. 335-342

[92] Ghahramani Z. Unsupervised learning. Vol. 3176. In: Bousquet O, von Luxburg U, Ratsch G, editors. Advanced Lectures on Machine Learning, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer-Verlag; 2004. pp. 72-112

[93] Duda RO et al. Unsupervised learning and clustering. In: Pattern Classification. 2nd ed. New York: John Wiley and Sons; 2001. pp. 517-600. ISBN: 0-471-05669-3

[94] Hinton G, Sejnowski TJ. Unsupervised Learning: Foundations of Neural Computation. Cambridge: MIT Press; 1999. ISBN: 0-262-58168-X

[95] In-memory MapReduce [Internet]. 2017. Available from: https://ignite.apache.org/features/mapreduce.html

[96] Apache HBase [Internet]. 2017. Available from: https://hbase.apache.org/

[97] Apache Hive [Internet]. 2017. Available from: https://hive.apache.org/. 2015

[98] Satish N et al. Designing efficient sorting algorithms for manycore GPUs. In: IPDPS 2009 IEEE International Symposium on Parallel & Distributed Processing. IEEE, IEEE XPlore Digital Library; 2009. pp. 1-10

[99] He B et al. Mars: A MapReduce framework on graphics processors. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. New York: ACM; 2008. pp. 260-269

[100] Mihailescu A. Stepping up theoretical investigations of ultrashort and intense laser pulses interacting with overdense plasmas. Combining particle-in-cell simulations with machine learning and big data. In: Proceedings of Grid, Cloud & High Performance Computing in Science (ROLCG), Conference. IEEE, IEEE Xplore Digital Library; 2015. p. 7367424

[101] Mihailescu A. A new approach to theoretical investigations of high harmonics generation by means of fs laser interaction with overdense plasma layers. Combining particle-in-cell simulations with machine learning. Journal of Instrumentation. 2016;**11**:C12004

[102] Apache Oozie [Internet]. 2017. Available from: http://oozie.apache.org/

[103] Azkaban Workflow Engine [Internet]. 2016. Available from: https://azkaban.github.io/

[104] Luigi Workflow Engine [Internet]. 2016. Available from: http://luigi.readthedocs.io/en/stable/api/luigi.contrib.sge.html

[105] Airflow Workflow Engine [Internet]. 2016. Available from: https://airflow.incubator.apache.org/

[106] Kepler [Internet]. 2016. Available from: https://kepler-project.org/

[107] Apache Yarn [Internet]. 2017. Available from: https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html

[108] Docker [Internet]. 2017. Available from: https://www.docker.com/

[109] Apache Tez [Internet]. 2017. Available from: https://tez.apache.org/

[110] Bulanov SV et al. Interaction of an ultrashort, relativistically strong laser pulse with an overdense plasma. Physics of Plasmas. 1994;**1**:745-757

[111] Brunel F. Not-so-resonant, resonant absorption. Physical Review Letters. 1987;**59**:52-55

[112] Kruer WL, Estabrook K. JxB heating by very intense laser light. Physics of Fluids. 1985;**28**:430

[113] Quere F et al. Coherent wake emission of high-order harmonics from overdense plasmas. Physical Review Letters. 2006;**96**:125004

[114] Hornik K. Approximation capabilities of multilayer feedforward networks. Neural Networks. 1991;**4**(2):251

[115] Rumelhart DE et al. Learning representations by back-propagating errors. Nature. 1986;**323**:533

[116] LeCun Y et al. Efficient BackProp. In: Orr G, Muller K, editors. Neural Networks: Tricks of the Trade. Berlin/Heidelberg: Springer; 1998

[117] Rumelhart DE, Zipser D. Feature discovery by competitive learning. Cognitive Science. 1985;**9**(1):75-112

[118] Ahalt S et al. Competitive learning algorithms for vector quantization. Neural Networks. 1990;**3**(3):277-290

[119] McCaffrey J. Test-Run, L1 and L2 regularization for machine learning, Microsoft Magazine, Issues and Downloads [Internet]. 2015. Available from: https://msdn.microsoft.com/en-us/magazine/dn904675.aspx

[120] Zou H, Hastie T. Regularization and variable selection via elastic net. Journal of the Royal Statistical Society. 2005;**67**(2):301-320

[121] Zeiler M. ADADELTA: An adaptive learning rate method [Internet]. 2012. Available from: https://arxiv.org/abs/1212.5701

IntechOpen