CHAPTER 1

Machine Learning for Monte Carlo simulations

COMPUTING techniques loosely based on mimicking the behavior of the human brain are becoming more and more important in a vast range of applications. Their utilization is not new, with studies based on simple Neural Networks (for example, [**?**, **?**, **?**]) dating back at least to the 80s; what is instead quite recent is the possibility to deploy efficient computing architectures, often specifically built to be optimal for such tasks.

At the same time, the capability to deploy larger and larger system has triggered theoretical studies, driving to more solid bases and to the definition of more complex and specialized models.

In these chapter we will start with an introduction to the models most relevant for Monte Carlo simulations, followed by a selection of applications. In the last part of the chapter, we will review strong and weak points about the utilization of Neural Networks applications for Monte Carlo simulations.

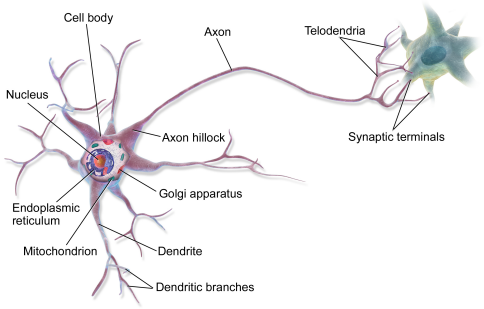
1.1 INTRODUCTION TO NEURAL NETWORKS

Neural Networks are a specific branch of the Artificial Intelligence (AI) domain in Computer Science. They get their inspiration from the fact that humans are evidently able to fulfill complex tasks; hence, by replicating the low-level mechanisms of the human brain on computing systems, one can potentially construct high level algorithms with similar capabilities.

1.1.1 The human brain

Neglecting any functional description, the human brain can be described as an organ composed by neurons, glial cells, neural stem cells and blood vessels (Figure 1.1a). In our current understanding, the neurons are the units

(a) A pictorial view of the human brain (from Wikipedia, by Patrick J. Lynch, CC BY 2.5).



(b) The human neuron (from Wikipedia, by Bruce Biaus, CC BY 3.0).

performing basic "operations" within the human brain, and their aggregated response is responsible for the high-level behaviour typical of humans. A neuron, as sketched in Figure 1.1b, is composed of three main units: a number of dendrites, the soma (the cell body), and an axion; the total size largely varies between different types of neurons; the neurons used for cognitive functions (as those in the grey matter of the brain) are usually short, of the order of 10 mm [**?**]. Functionally, a neuron is able to generate an electric response on the axion (*output*), depending on the electrical potential present at the synapses (*inputs*) on the dendrites. Neurons can be chained by connections between axions and dendrites, generating a configuration in which N neurons are connected via M synapses. The high-level response of the human brain to stimuli is understood to come from the complexity of such connections, with a standard human brain featuring $\sim 10^{11}$ neurons each with 7000 synapses, for a total of $\sim 10^{15}$ synapses; as we will see in the following, the brain can be interpreted as a complex mathematical system with 10^{15} degrees of freedom.

In literature various models of the neuron behavior have been proposed [**?**, **?**]; here, we will focus on the simplest yet most simple to implement in computer systems [**?**] (see Figure 1.2): in this model, the *output* y signal at

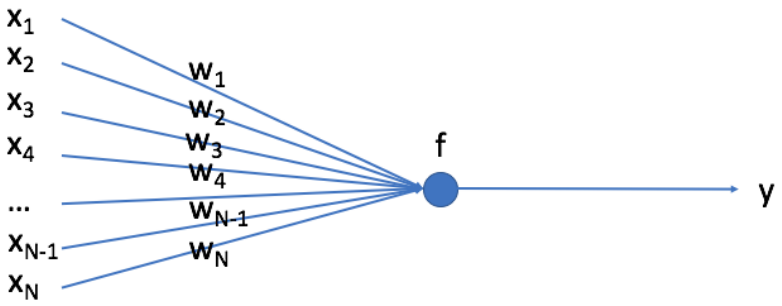


FIGURE 1.2: The artificial neuron.

the axion is assumed to be a function of the *inputs* x_i via

$$y = f\left(\sum_{i=1}^{N} w_i x_i\right) \tag{1.1}$$

where w_i are weights modeled after the chemical potentials at the synapses, and the function f wants to model the non linearity of response of biological neurons with the *inputs*; on top of this, the function f is needed in the mathematical model in order to allow the description of non linear phenomena [**?**]. The perceptron [**?**], one of the first models used in literature to model Neural Networks, uses a very similar model, with a simplified f function which is simply

$$f(\vec{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

In modern systems, two modifications are typical when using Neural Networks:

- the addition of a further synapse x_0 which is always 1, used as a bias to the system; its weight is referred to as x_0 or b (as in *bias*).
- the use of continuous f non linear functions, as logistic or hyperbolic [**?**] functions, in order to model non binary signals.

Neural networks are designed by combining multiple neurons in *networks*, usually in a layered structure: one layer is used to map the inputs, a few/many layers are *hidden*, and a single layer is used to to map the outputs. On top of that, more complex neurons can be used, for example including a "memory" cell, or presenting a recurrent behavior by reusing its output as one of the inputs. A full description of all the type of neurons and networks is beyond the scope of this chapter; in the following, the ones most relevant to Monte Carlo simulations will be presented with more detail. For reference, still, a

quite complete classification of currently relevant neural networks is shown in Figure 1.3.

As visible there, some network topologies have a high number of hidden layers. While the Universal Approximation Theorem [**?**] states that, under quite generic conditions, a single hidden layer between inputs and outputs should be enough in all cases, networks used in science during the last decade tend to be "deep" (i.e. with many hidden layers) [**?**]. This has multiple motivations: on one side, the theorem states that it is possible to have just one hidden layers, but does not state with how many neurons (and it tends to be a very large number); on the other side, a deep structure tend to be better human readable, with cascade sub networks with identifiable and logical roles. Hence, relevant networks in today's science tend to be deep.

The typical utilization pattern for a majority of network topologies is to feed them as input a large set of data representing the problem of interest, be it a medical image, a set of features or any output from the instrumentation, and at the same time provide the "expected output" from a so-called training set.

In the simplest type of networks, the responses from the neurons in the hidden and the final layers are considered in sequence (inputs to outputs), with each layer *feeding* the following layers; hence the name Feed Forward Neural Networks (FFNNs).

During the training, the network adjusts its internal free degrees of freedom (the weights w_i^j in equation 1.1, extended to the j neurons in the various layers) to better reproduce the desired answers, via minimization procedures which can be either numerical or analytical and involve the definition of a loss function to be explicitly minimized. Typical loss function can be simple weighted differences between the networks results and "expected results", but functional forms like cross entropy and mean square errors are more typical [**?**].

What has just been described is the training process for *supervised* Neural networks, which rely on an externally provided "truth" to adjust for optimal performance, without having any a-priori knowledge or description of the physical process they want to reproduce. Other topologies describe instead *unsupervised* Neural Networks, in which the training process just implies the utilization of datasets without the need to provide the correct answers (which can be unknown). Examples of such networks will be provided in Section 1.1.4.1.

1.1.2 Convolutional networks

Convolutional networks (CNNs) are a useful subset of neural networks, which exhibit peculiar characteristics of showing response space invariant with respect of the inputs.

CNNs use basic neurons as explained in Section 1.1.1, with those in the hidden layers fed by small portions of the inputs per iteration, thus realizing spatial independence. As depicted in Figure 1.4, multiple application of several convolutional layers and pooling layers drive to an overall analysis on the

inputs, and to the final outputs. Inputs can be either one dimensional (e.g. vectors of features), two dimensional (e.g. images) or multidimensional arrays. CNNs are used with success in categorization problems, where specific structures, must be discovered into a set of input data: a typical example is the identification of images with lesions in medical imaging, where CNN classifiers are trained on reported images by clinicians [**?**].

CNNs are particularly interesting in the realm of Monte Carlo simulations, since the space invariance is a valuable characteristics: the energy deposition of a particle into a material does not depend on the specific entry point nor on its direction.

1.1.3 Recurrent networks

The CNNs above described are a type of *stateless* network, in which the output depends only (given a static set of weights) on the inputs presented on the first layer. In Recurrent Neural Networks (RNNs), the response to a specific input feature set also depends on the history of the network, i.e. the inputs presented at prior times; as such, the network presents a "memory", which can be used to correlate multiple inputs in sequence. A typical utilization pattern is in the presence of inputs of variable length, which cannot be estimated *a priori*, like in the analysis of text or speech sequences. In simulation processes, the feature is often utilized when the signals from a non fixed number of inputs (i.e. the incoming particles to a volume) must be piled up in a coherent way: the inputs per each particle are presented to the network, with a specific input pattern which may or may not be used to signal the end of the sequence.

1.1.4 Generative Models

The networks popular up to 10 years ago were mostly useful during decision processes, such as categorizing inputs (signal vs background, for example) or counting and defining specific regions inside it (segmentation, counting of lesions, etc).

In order to be applied to Monte Carlo simulations, instead, the capability to produce ("generate") an output as close as possible to reality, or to a more standard algorithm is essential. In order to do this, different network topologies and strategies for training are relevant.

1.1.4.1 Auto-Encoders and Variational Auto-Encoders

The autoencoders are a family of unsupervised Neural Networks designed to learn a lower dimensional representation (say N dimensions) out of a set of data (with M dimensions, M>N). The N dimension representation ("latent space") can be seen as coming from an "understanding" of internal patterns and correlations in the initial data.

The easiest form of autoencoder has the number of inputs and outputs

equal to M, and an internal hidden layer at dimensionality N. The training is obtained by forcing the network during training to reproduce as close a possible the input features at the output layer, thus requesting the N-dimension space to be as perfomant as possible when representing the M-dimension inputs (see Figure 1.5).

Autoencoders in such form are used for two distinct purposes:

- auto-discover in the inputs hidden symmetries or underlying correlations, which can be used, for example, in lossy compression [**?**] or to drive understanding on the inputs themselves;
- since the network is trained on a specific data sample, it will minimize the difference outputs vs inputs on that specific dataset. Once the same network is presented with "different" data, it is expected to fail to reproduce the inputs at the outputs; hence, it can be used to detect anomalous inputs, such as corrupted input samples in medical imaging analysis pipelines [**?**] or not expected events in High Energy Physics [**?**].

Auto-Encoders are only able to reproduce the elements of the training set but not to interpolate among them; Kingma and Welling [**?**] introduced the Variational Auto-Encoder (VAE) class of algorithms, or more correctly "Auto-Encoding Variational Bayes" methods as they are more related to Variational Bayesian methods then to Auto-Encoders. The main differences of VAEs with respect to AEs is that VAEs encodes the input into a probability density function (PDF) rather than a point in the latent space; in this way, close points in the latent space generate events similar to each other. This is essential if willing to use VAEs as generators, as in Monte Carlo simulations: a sampling of the latent space "close" to the training configurations can be used to generate "valid" output configurations. In standard AEs, instead, only the precise points in the latent space where the the training set inputs have been encoded can be safely used to generate outputs.

As said, VAEs, once trained to generate elements from points sampled nearby the latent space position where the training set events are encoded, can be used to *generate* realistic data, interpolating or extrapolating from the training set events; this make them very useful for the generation of Monte Carlo simulations.

The loss functions of AEs is a "distance" between the input element and the one generated in output; in VAEs there is another term, typically a Kullback-Leibler (KL) [**?**] divergence, that measures the distance between the encoding PDF and a reference PDF, usually a Normal distribution.

There is no guarantee that the latent space gets optimally organized with respect to the input features. However, it has been shown [**?**] that increasing the coefficient of the KL term in the sum of the two loss function addenda, increases the probability that the VAE learns a disentagled representation of the input. This hyper parameter is usually referred as β and this modification of the VAE, β-VAE.

VAEs tend to produce outputs blurred with respect to inputs, however, they have the great advantage that it is possible, especially with β-VA, to control the features of the generated output and to interpolate with continuity between two input samples.

1.1.4.2 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a recent [**?**] class of networks designed to reproduce the behaviour of complex systems without an explicit programming.

The method (see Figure 1.6) implies putting in competition a reference algorithm (or real data, A) with a generative network (B), which produces an output with the same structure and whose response, initially, is sampled from a random space. A second network (C) is trained on the capability to distinguish between the algorithm's output and the generated one. The two networks are put in competition (hence the term *adversarial*), with B "winning" if it can convince C that its response is the real one; in the opposite case, C wins. The tension between the two networks pushes the network B to generate outputs which are indistinguishable from the real data / the output of the algorithms. B and C are generic networks, with convolutional neural structures mostly used.

GANs are more and more used when in the need for replicating the behaviour of a known data source, with improved computing performance. Upon successful training, in fact, the network B has the typical speed of (say) a CNN, and is able to reproduce the output of A, which can be a complex and time consuming algorithm. Successful examples are the famous generation of realistic faces [**?**], the capability to reproduce the showering of particles in calorimeters [**?**], or for for beam source modelling in Radiation Therapy linacs [**?**].

1.1.4.3 Graph Networks

In a quite general category of problems, the task is to discover the relations between objects; in the segmentation of medical images, for example, there is the need to understand whether certain areas are part of a specific organs. Clusterization algorithms in many science realms is another example [**?**]. The most used network architecture for these use cases is a Graph Neural Network (GNN, [**?**]). In a GNN, nodes (representing objects) and edges (representing connections between nodes) are the basic entities, and the training process aims to correctly categorize the nodes, via an assessment of the strength of each connection. GNNs are finding large applications in cases where one wants to replace a complex, combinatorial algorithms with a Neural Network: typical examples in literature are jet clustering in High Energy Physics [**?**], and tracking in dense particle environments [**?**].

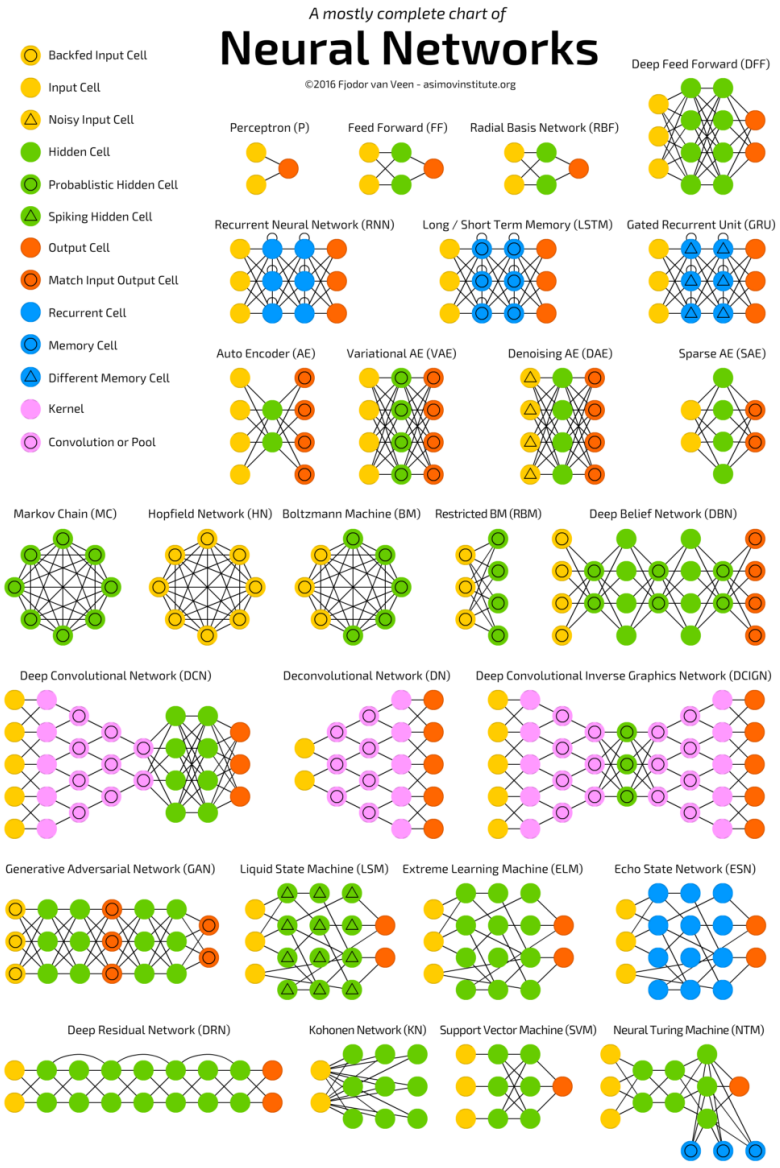


FIGURE 1.3: Types of neurons and neural networks currently relevant in literature (Copyright F. van Veen 2016).

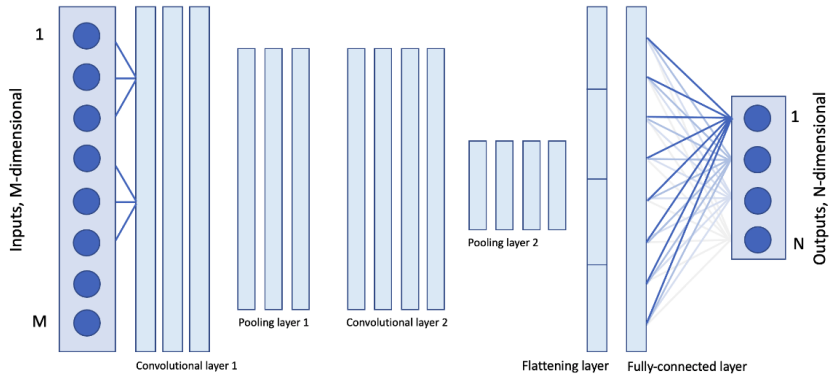


FIGURE 1.4: Basic structure of a CNN.

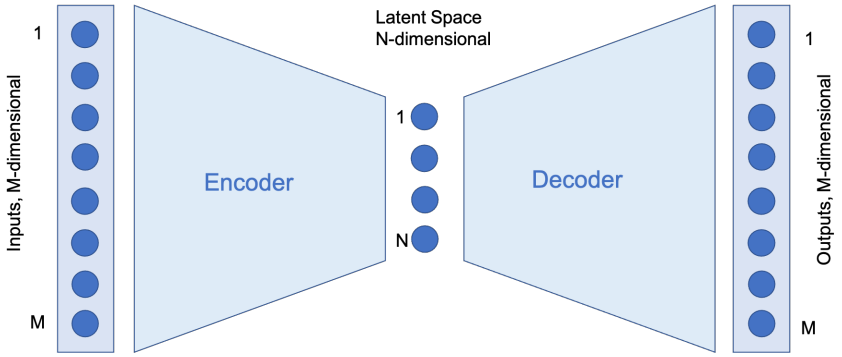


FIGURE 1.5: An autoencoder in its simplest form.

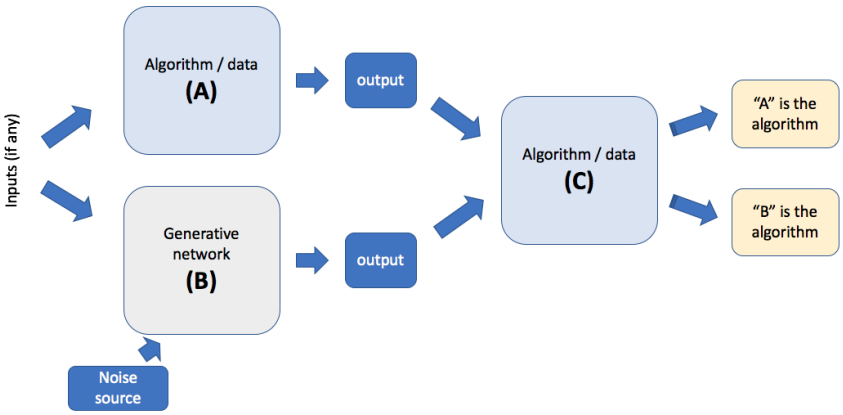


FIGURE 1.6: The structure of a simple GAN.

1.2 EXAMPLES OF APPLICATIONS FOR MONTE CARLO SIMULATIONS

1.2.1 Emulation of radiation-matter interactions

One of the most complex tasks in Monte Carlo simulations involving the use of detectors (medical apparatuses, particle physics detectors) lies in the dual need of being able to optimize the design before detector construction, and to simulate the behavior under working conditions after the setup has been prepared. In both cases, unless the setup is very similar to existing detectors, extensive Monte Carlo simulations of the expected detector capabilities are the widely used solutions. Various such tools exist (Geant4 [**?**], Fluka [**?**], MCNPX [**?**]), with different application regimes and specific utilization patterns. As a general rule, these implement iteratively basic radiation-matter low-level processes to a knowledge of the detector setup, including materials, geometry and other experimental conditions; as such, they incur into two general limits:

- a scarce capability to be tuned to experimental results, by changing the basic modelling of the processes;
- a large to very large need for computational resources, given the iteration oriented approach and the need to increase the level of iterations in order to obtain a better precision and adherence to data.

Both limitations can in principle be surpassed via the use of Artificial Intelligence oriented tools. In presence of experimental data, the response of the AI system can be tuned to that without any explicit modelling of the physics processes; speed can be vastly improved by the change from iteration-based computations to standard Deep Learning matrix algebra, with its intrinsic capabilities for high performance processing on, for example, GPU systems.

As an example, we want to consider here CaloGan [**?**], an attempt to reproduce the details of radiation-matter interactions in the complex setup of segmented (3 layers) electromagnetic calorimeters. A Generative Adversarial Network, as those presented in Section 1.1.4.2, is used in conjunction with an as-accurate-as-possible Geant4 simulation of the same experimental setup. The generator side accepts in input particles' 4-momenta, and, after the passage through quite standard convolutional (matching the detector response as 2-D images) and dense layers, the output is compared with detailed Geant4 simulations of particles with the same parameters. The training optimizes the energy deposition per layer and per 2-D transverse cell, in a way in principle suited also for using real data in input. Results are very encouraging, even in an extreme detector scenario: not only the quantities of direct training are well reproduced, but also secondary and derived quantities like shower shapes are in most cases well described.

Results are shown in Figure 1.7 for the explicit targets of the calculation (2-D layered images of the energy deposits, in the specific case of incoming positrons).

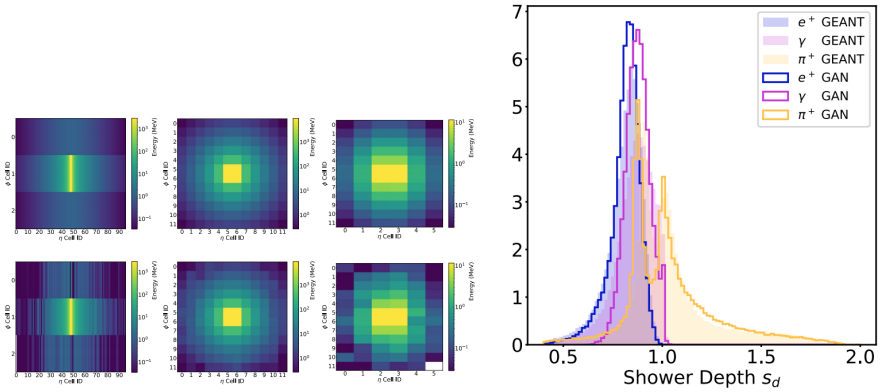


FIGURE 1.7: (left) Average e^+ Geant4 shower (top), and average e^+ CALOGAN shower (bottom), with progressive calorimeter depth (left to right). (right) Energy weighted shower depth (in a.u.) from CaloGan and Geant4 detailed simulations. (From [**?**]).

Interestingly, one can look into quantities derived from shower shapes, but not directly targeted by the GAN training step, like for example the energy weighted shower depth. As we will see in Section 1.3.1, in general one cannot assume these quantities will be perfectly reproduced; in this specific case, though, the agreement level is quite impressive.

A second similar attempt, applied to the not-yet existing CLIC proposed electromagnetic and hadronic calorimeter, is presented in [**?**], with the goal to directly reproduce 3-D signals in a highly granular calorimeter. The reference dataset, in absence of real data, has the form of Geant4 generated showers sampled in a 25x25x25 cells around the impinging particle. Figure 1.8(left) shows the longitudinal shower shapes for 100 GeV electrons in the electromagnetic part of the calorimeter compared with detailed Geant4 simulations. The level of agreement is very satisfactory. Figure 1.8(right) shows a pictorial rendering of the expected energy deposit by a 100 GeV electron in the calorimeter.

In sections 1.3.1 and 1.3.2 we will discuss about the speed gain with respect to standard methods, and solutions and needs to prevent unphysical results.

The last two examples are related to High-Energy particle Physics, however, a similar approach can be used also to emulate the energy deposition in a voxel geometry, such as the CT of a patient.

In the field of hadron therapy, strong requirements are imposed on the accuracy in predicting the range of the treatment field in patients, and the development of patient specific dose verification methods is highly desired. Positron Emission Tomography (PET) imaging can be used to monitor the activity generated in the body tissues by the interaction with the therapeutic

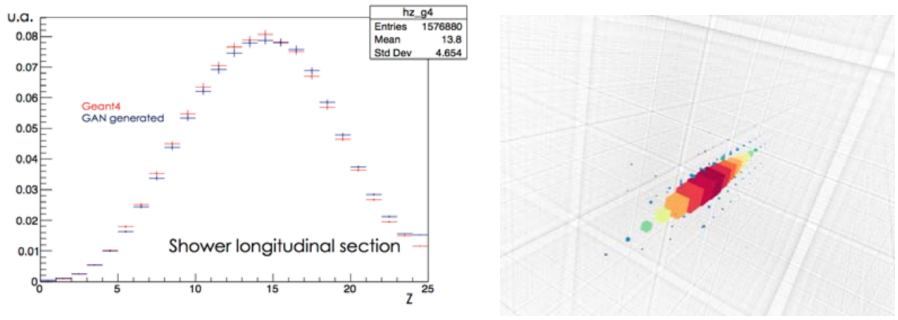


FIGURE 1.8: (left) Longitudinal shower shapes for 100 GeV electrons: GAN result is compared to full Geant4 simulation. The Z coordinate indicates the bin index in the longitudinal direction. (right) The three-dimensional representation of an energy shower created by a 100 GeV electron as generated by the GAN, using particle type as conditioning information. (From [**?**]).

hadronic beams. Attempt to establish the correlation between the activity measured with a PET detector and the dose released to tissues have been carried out by utilizing machine learning techniques based on RNN in the study by Hu et al. [**?**] and on GAN in the study by Ma et al. [**?**]. Imaging the prompt gamma radiation is another approach to dose monitoring in hadron therapy. A deep learning based conversion of the prompt gamma information into proton dose distribution has been proposed by Liu et al [**?**].

1.2.1.1 Emulation of detector responses

Monte Carlo tools like Geant4 are designed to simulate, as accurately as possible, the energy deposition (in keV, for example) due to the passage of particle / radiation in the material of a detector. In real life, what a scientist measures is instead the response, as analog or digital signals, of a measuring device in which energy deposition is read and processed by some electronic boards. In classical systems, the simulation of radiation / matter must be followed by an ad-hoc simulation of the electronic readout system, in order to be compared with actual readings from a detector. In the case of AI inspired tools, this can be avoided by completely bypassing the "energy deposition" output results, and training the system directly with real or realistic (from the above ad-hoc simulation) signals from the electronic back-end (see Figure 1.9).

MC simulations are also frequently used to estimate the efficiency and the geometrical acceptance of a detector, or a system of detectors. The simulation of a Single Photon Emission Computed Tomography (SPECT) is usually done in two steps: the first one simulates the interactions of the produced photons with the patient, producing as output the emerging photons; and the second one simulates the response of the collimator-detector system to the emerging

photons. This second step is called Angular Response Function (ARF). Sarrut et al. [?] used a GAN to emulate the ARF. The DL algorithm takes as input the incidence angles of the photon and its energy and gives back the probability of detecting it in one of the possible energy windows. In summary, the DL algorithm emulates the detector system, including the collimators.

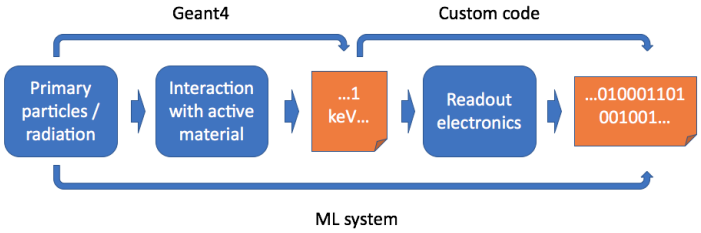


FIGURE 1.9: Difference between classical and AI inspired simulation of experimental setups.

1.2.2 Emulation of nuclear interaction models

Nuclear reactions models are one of the most demanding part of a MC simulation in terms of running time. Despite the large running time, the models already available in toolkits made to develop MC simulation, such as Geant4, have shown severe limitations in reproducing experimental data below 100 MeV/u [?]. Models developed by theoreticians for this energy domain can be interfaced with Geant4 with good results [?], however their running time is even larger. Ciardiello et al. [?] obtained encouraging preliminary results in emulating one of the state-of-the-art models for low energy nuclear reactions with a VAE. The model in question is BLOB ("Boltzmann-Langevin One Body") [?], which simulates the first part of the nuclear reaction, from the contact of the two nuclei until the energy of the nucleons composing the fragments is balanced among them. The BLOB output is a PDF of finding a nucleon in a given position of the phase space. The authors trained a VAE in reproducing the BLOB prediction in the interaction of two ^{12}C nuclei at 62 MeV/u. For this purpose, they discretized and reduced the dimensionality of the BLOB output to use 3D convolutional layers. In detail, the dimensionality reduction uses the fact that in the reaction in exam BLOB predicts at most three large fragments, i.e. larger than one nucleon. Therefore, they divided the PDF produced by BLOB in three PDFs, one per large fragment, and associated all the nucleon emitted in the first part of the reaction to one of these three large fragments. In this way they used the three colour channels of convolutional layers to represent each of the possible large fragments. In spite of controlling the generative part, they trained a classifier for the event impact parameter (b) jointly with the VAE itself. This joint training helps the VAE in learning a the task, given the large sparsity of the input, and

force the latent space in being organise with respect to the impact parameter. Moreover, it will be possible to sample from the latent space deciding the impact parameter of each generation. Figure 1.10 shows that the VAE is able to generate PDFs very similar to the input one when sampling a point nearby the position in the latent space where the input has been encoded.

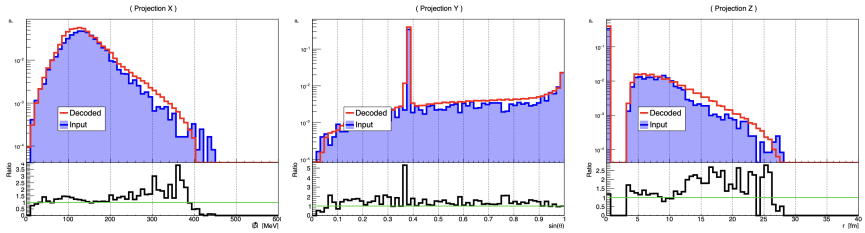


FIGURE 1.10: Example of results obtained in generating with the VAE a distribution, in red, starting from a point sampled from the neighborhood of the point where the input distribution, in blue, is encoded. The three distributions are the projections on the three axis of 3D PDFs.

1.2.3 A possible application of Graph Neural Networks

To date, no specific study on Monte Carlo simulations of radiation / matter interactions using Graph Networks can be found in literature; still, it is clear from studies in other scientific domains suggest the potential of the technique. We want to cite here a study simulating the mechanical behavior of a fluid system, which shows capabilities of rendering the physics behind a complex system such as an ensemble of water small volumes [**?**]; we expect similar GNN systems to become available for our research fields in short time.

1.3 STRONG AND WEAK POINTS ABOUT NEURAL NETWORKS APPLICATIONS FOR MONTE CARLO SIMULATIONS

Different AI based techniques for Monte Carlo simulations have been presented in literature, as shown in Section 1.2; their level of maturity varies between proofs of concept, advanced tests and production systems. Still, while we may be convinced that Neural Networks can be a reasonable substitute for classical algorithms, that would not be enough in absence of clear advantages in other areas.

1.3.1 Speed, accuracy and reliability

As described in Section 1.2.1, one of the main expected advantage of AI inspired systems, when compared to more standard simulation algorithms, is a potential speed-up without necessarily impacting performance. The improvement comes from the nature of feed-forwards Neural Networks used at inference time, which are described as a series of matrix multiplication of fixed length[1], without access to any data structure apart from the inputs and the weights. Today's processors, from CPUs to GPUs to TPUs, are very efficient in processing matrix algebra calculations; this reflects directly to short processing times. In order to give quantitative examples, we can refer to the two complex GANs described in Section 1.2.1, which are fed by Geant4-processed events in the training phase. Table 1.3.1 reports on the absolute time needed to process Geant4 vs the GAN for events with similar input; the speedups are of order 100x using CPUs, and an additional factor 100x with GPUs[2].

System	Software	Speed
Intel Xeon E5-2683	Geant4	1 min
Nvidia RTX 1080	3d-GAN	0.04 msec
AWS `p2.8xlarge`	Geant4	1.7 sec
AWS `p2.8xlarge`	CaloGan	O(10) msec
AWS `p2.8xlarge` + Nvidia K80	CaloGan	O(0.01) msec

Fig 1.3.1: Geant4 vs GAN performance under various setups, as extracted from [**?**] and [**?**].

So, speed-wise there is a clear advantage; but what about the accuracy of the simulations? We can refer back to Figures 1.7 and 1.8 to compare the reference (detailed simulations) with AI predictions. In general, a perfect agreement is not to be expected, but in many applications a O(10)% agreement at the much reduced cost is a viable solution, also because even the detailed simulation is not expected to be perfectly describing the data. A different problem is present

[1] This is not strictly valid for some types of recurrent networks, where the recurrence can introduce an indetermination in the sequence of mathematical operations.

[2] It can be said that it is an unfair comparison since Geant4 cannot currently use GPUs; still, it shows how AI inspired tools offer a path to the use of more performing technologies.

when one wants to use the AI system beyond the quantities explicitly used in the training phase: while these can be accurately reproduced, there is no guarantee that derived or different quantities are, since these are not explicit target of the minimization procedure. In these case, as discussed in [**?**], an accurate check should be done *a posteriori* on all the quantities used from the system; if any of them turns out to be unsatisfactory, the standard solution is to include them explicitly in the GAN training as an additional target.

An item which recently has gained a lot of attention is the capability to explain results from Machine Learning systems: the high number of degrees of freedom combined with the non-linearity of response makes to difficult to justify results from first principles. While this is not a priori a problem in most applications, it becomes worrying when mission critical and potentially dangerous systems are driven by AI decisions (think of treatment plans for radiotherapy, or the assessment of radiation damages in industrial environments). Explainability of ML results is not a solved problem; still, tools and procedures exist in order to identify typical problems and effects. This is beyond the scope of this chapter, and a review can be found in [**?**].

1.3.2 Unphysical responses and how to impose physical constraints

AI inspired tools are a mathematical solution to problems we have problems solving algorithmically, either because too difficult or too slow. An important difference with respect to standard "human-written" code is the impossibility to impose at algorithmic level precise conservation laws, such as the conservation of energy and momentum. There is no guarantee a ML system will conserve them, since their validity is not implemented in any explicit form but should be recognized as an emergent behavior of the system.

In most of the cases shown in Section 1.2 the strict conservation (for example) of energy in particle showering into calorimeters does not need to be exact, since in the transfer between the energy in the impinging particle and the final products is in any case approximate due to effects of leaking, and smoothed by the detector resolution. In cases like this, while the training tries to match the "cell by cell" energy deposition, there is no explicit request that the total energy deposited in the calorimeter matches the precise simulation. In principle, a loss function designed to have the minimum when all the cells have the "expected" energy would suffice, but experience (and somehow common sense) shows that by adding to the loss function an explicit term tending to the conservation of total energy, like in Eq. (2) in [**?**], constrains can implemented even if not in exact form. This solution is called "soft" precisely due to this. The relative weight of the various terms in the loss functions are somehow arbitrary, and an higher weight to the part tending to the constraint (not really imposing it!) represents the developer desire to have it more precisely maintained, even if never exactly.

A different approach, as presented in [**?**], tries to implement instead "hard" and exact constraints on the outputs during the training phase; though, the

mathematical complexity and the time needed increase, such as to advice the use of (eventually up-weighted) "soft" constraints in any case.