Universidade Federal de Alfenas

Tiago Vitor Lopes

SUPERVISED NEURAL NETWORK APPROACH TO MODELING DUNE'S LArTPC PHOTON DETECTOR DEVICE

Poços de Caldas/MG

2023

Tiago Vitor Lopes

SUPERVISED NEURAL NETWORK APPROACH TO MODELING DUNE'S LArTPC PHOTON DETECTOR DEVICE

Dissertation submitted to the Graduate Program in Physics at the Universidade Federal de Alfenas - MG, Brazil, as part of the requirements for obtaining the Master degree in Physics. Research area: High-Energy Physics; Machine Learning.
Advisor: Dr. Anibal Thiago Bezerra.

Poços de Caldas/MG

2023

Sistema de Bibliotecas da Universidade Federal de Alfenas
Biblioteca Campus Poços de Caldas

Lopes, Tiago Vitor.
Supervised Neural Network Approach to Modeling DUNE's LArTPC Photon Detector Device / Tiago Vitor Lopes. - Poços de Caldas, MG, 2023.
61 f. : il. -

Orientador(a): Anibal Thiago Bezerra.
Dissertação (Mestrado em Física) - Universidade Federal de Alfenas, Poços de Caldas, MG, 2023.
Bibliografia.

1. Machine learning. 2. Neural networks. 3. Dune experiment. 4. High energy physics. I. Bezerra, Anibal Thiago, orient. II. Título.

Ficha gerada automaticamente com dados fornecidos pelo autor.

TIAGO VITOR LOPES

SUPERVISED NEURAL NETWORK APPROACH TO MODELING DUNE'S LArTPC PHOTON DETECTOR DEVICE

O Presidente da banca examinadora abaixo assina a aprovação da Dissertação apresentada como parte dos requisitos para a obtenção do título de Mestre em Física pela Universidade Federal de Alfenas. Área de concentração: Matéria condensada

Aprovada em: 20 de dezembro de 2023.

Prof. Dr. Aníbal Thiago Bezerra

Presidente da Banca Examinadora

Instituição: Universidade Federal de Alfenas

Prof. Dr. Eric Batista Ferreira

Instituição: Universidade Federal de Alfenas

Prof. Dr. André Fabiano Steklain Lisboa

Instituição: Universidade Tecnológica Federal de Paraná

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Wir müssen wissen. Wir werden wissen.

- David Hilbert (42)

Every farewell is a death

In the train we call life

We are all chance events in one another's lives

And we all feel sorry when it is time to get off.

Fernando Pessoa (43)

ABSTRACT

This work presents the development and evaluation of an artificial neural network (ANN) as a supervised learning model to complement the ArapucaSim software simulation of the behavior of Arapucas' photon absorption probabilities on DUNE. DUNE will be a neutrino detector intended to address fundamental questions about the nature of elementary particles and their role in the universe. The Arapucas are the light-trapping devices proposed for the detection system of the DUNE far detector. A neural network model employed is a regressor model that receives as inputs the coordinates of a photon generator, along with positions on the surface device where the photon collides and produces outputs consisting in absorption probability for each photon. The input data is obtained from Geant4 simulations, specifically from the ArapucaSim module. Two cases were studied: one in which the photons arrive with normal incidence in the Arapuca surface, and the other including the angle dependence. While the Geant4 approach required hours to generate results, after training the neural network model, comparable probabilities are produced in seconds with high accuracy. This work shows the potential of neural network models as efficient alternatives for simulations in predicting photon absorption probabilities by the light sensor with less time and computational effort.

Keywords: machine learning; neural networks; Dune experiment; high energy physics.

RESUMO

Este trabalho apresenta o desenvolvimento e avaliação de uma rede neural artificial (RNA) como modelo de aprendizagem supervisionada para complementar a simulação do software ArapucaSim do comportamento de Arapucas, os dispositivos de captura de luz utilizados no Experimento DUNE, para modelar probabilidades de absorção de fótons. DUNE será um detector de neutrinos destinado a abordar questões fundamentais sobre a natureza das partículas elementares e o seu papel no universo. O Arapuca é o dispositivo proposto para o sistema de detecção do detector distante (Far Detector) do DUNE. O modelo de rede neural empregado é um modelo regressor que recebe como entradas as coordenadas de um gerador de fótons, juntamente com as posições no dispositivo de superfície onde o fóton colide e produz saídas que consistem na probabilidade de absorção para cada fóton. Os dados de entrada são obtidos a partir de simulações Geant4, especificamente do módulo ArapucaSim. Foram estudados dois casos: um em que os fótons chegam com incidência normal na superfície de Arapuca e outro incluindo a dependência angular. Embora a abordagem Geant4 exija horas para gerar resultados, após treinar o modelo de rede neural, probabilidades comparáveis são produzidas em segundos com alta precisão. Este trabalho mostra o potencial dos modelos de redes neurais como alternativas eficientes para simulações que prevêem probabilidades de absorção de fótons pelo sensor de luz com menor tempo e esforço computacional.

Keywords: aprendizado de máquina; redes neurais; Dune experiment; high energy physics.

LIST OF FIGURES

TABLE OF CONTENTS

1 INTRODUCTION

This work explores the use of artificial neural networks (ANNs) to model the photon absorption probabilities in the Arapuca light-trapping devices that will be used in the Deep Underground Neutrino Experiment (DUNE). DUNE aims to study fundamental neutrino properties using large liquid argon time projection chambers (LArTPCs) containing tens of kilotons of pure liquid argon (LAr). Critical to reconstructing these complex events and particle trajectories is DUNE's photon detection system to capture scintillation light from liquid argon. Details on how the DUNE Experiment works and its primary goals are on Section 2.3.

When charged particles pass through the LArTPCs, they ionize argon atoms, emitting scintillation light in the ultraviolet region at 128 nm. Efficiently detecting these argon scintillation photons is crucial for reconstructing three-dimensional event topologies and energy depositions in the LArTPCs. The Arapuca is a novel technology proposed for DUNE's photon detection system, designed to achieve wide-area coverage and detection efficiency around 2.5% by trapping wavelength-shifted photons inside a dichroic filter box until they are absorbed by a silicon photomultiplier (SiPM). Section 2.3.1 deals with the Arapucas, a family of photon detectors developed in Brazil with the current version used in DUNE's detector being the X-Arapucas.

The current simulation of the X-Arapuca is done with a software called ArapucaSim (Section 4.1) using Monte Carlo methods. While Monte Carlo simulations using software like Geant4 can accurately model the optical processes and probability responses of Arapucas, these simulations are computationally expensive, time-consuming, and inefficient to be integrated within the simulation chain. An alternative approach considered to be taken to facilitate the modeling of the light absorption by the Arapucas are state-of-the-art Machine Learning (ML) models.

Hence, this dissertation investigates using artificial neural networks as a faster surrogate model to predict probabilities for the Arapucas. Custom data sets consisting of Arapuca absorption efficiencies generated from the simulation with ArapucaSim are used to train neural network regressor models to rapidly output continuous probability values for new data based on the learned patterns. Different neural network architectures are tested as regressor models (Section 4.2). These are trained on data sets of absorption

efficiencies simulated under two conditions: normal incidence photons and with different photon incidence angles. The inputs consist of coordinates of the photon origin and their arrival points on Arapuca's surface, while the output is the probability of absorption by the Arapuca. Activation functions like rectified linear units (ReLUs) and regularization techniques like dropout are implemented to optimize the model performance. Various training parameters including learning rate, batch size, and number of epochs are tuned using at the same time methods like learning rate scheduling and early stopping to improve convergence (Section 4.2). Different loss functions like mean squared error (MSE) are evaluated as training objectives. The trained networks are tested on unseen simulated data and generate absorption probabilities that match the simulations.

The results presented on Section 5 demonstrate that neural network regressor models can rapidly produce accurate absorption probabilities compared to Geant4 simulations, providing a speedup of orders of magnitude in generating data.

2 THEORETICAL BACKGROUND

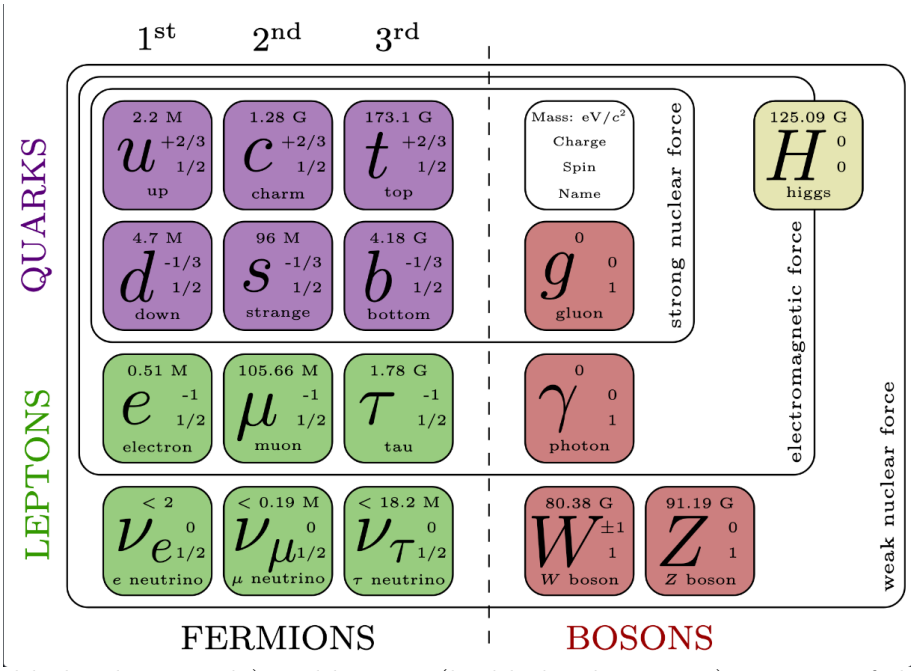
2.1 Elementary Particles

"What is matter made of?" is the question put forward by Elementary Particle Physics addressing the microscopic scale. The building blocks of the universe, as it turns out, consist of particles of different types: electrons, protons, neutrons, and many others, and all of them have their properties: mass, charge, spin, and others (Figure 1). Since these particles are too small, the models of their features and how they behave are probed in more dramatic ways than that of Classical Physics experiments, through scattering, decays and bound states. The construction of models, namely laws for the behavior of particles, relies particularly on Quantum Field Theory (QFT) using Feynman's framework. QFT is the combination of Special relativity (SR), which deals with fast-moving objects, and Quantum Mechanics (QM), which deals with small objects. SR allows for the existence of particles with zero mass and interaction in which rest mass is not conserved; Conjointly, QM allows for non deterministic behaviour of particles from initial conditions, that is to say, particles can decay or scatter from one state to another with a probability associated with it.

Two questions arise when elementary particles are studied: how does one produce them and how does one detect them? Electrons can be generated by heating a wire or metal plate; Ionizing hydrogen gives forth protons; High-energy particles coming from outer space hit atoms in earth's atmosphere creating secondary particles such as muons; Disintegrating nucleus emits alpha particles (Helium nucleus), beta particles (high energy electrons or positrons), gamma rays (high energy photons), neutrons and neutrinos; Colliding electrons and protons on targets such as blocks of graphite or beryllium in particle accelerators, generates secondary particles - pions, which decay into tertiary particles, such as neutrinos and muons. The smaller the range of the interaction studied, the higher the energy taken for colliding the particles, as expressed by the De Broglie relation $\lambda = h/p$ and the Uncertainty Principle $\Delta x \Delta p \geq \hbar/2$ (2).

On the other hand, detecting particles is basically done by tracking the particle's path once it ionizes a medium, being a gas or liquid inside a chamber, for example. The

Figure 1 – The Standard Model of particle physics representing fundamental particles and their properties.



Quarks (highlighted in purple) and leptons (highlighted in green) are part of the Fermions generations, displaying their masses, charges, and spins. Bosons (highlighted in red) are responsible for mediating fundamental forces. Source: (3)

chamber, although there are many other types of detectors, can be designed in such a way that the information collected about the interaction allows for its reconstruction in computer softwares. The present work describes the detection system in the Dune experiment, which is a Time Projection Chamber (TPC) filled with liquid argon as a sensitive medium that allows for 3D reconstruction of interactions. Those are specifically called LArTPC's and they were proposed in the 1970s to study neutrino-elecron scattering, solar or cosmic neutrinos and proton decay (37, 38, 39).

Let's briefly summarize the history of how particles were discovered: J.J. Thomson discovered the electron in 1897. He knew back then that cathode beams carried negative charge due to the direction in which they were deflected by magnets. He determined the velocity and the charge-mass ratio of the particles by passing them through adjusted electric and magnetic fields. He imagined the charges, which he called electrons, being distributed on top of a positively charged cloud to explain the atom structure, since it was heavier than the electron and neutral. In 1909, E. Rutherford discovered that this positively charged cloud was concentrated in the nucleus of the atom, after he conducted his experience firing alpha particles in a gold thin film and analysing the deflected angles

of these particles. He gave the name proton to the hydrogen nucleus. N. Bohr calculated the electromagnetic spectrum of hydrogen in 1914 using a planetary model of the atom, in which the proton in the center is orbited by one electron. Wrongfully so, the heavier atoms were then thought to have masses in multiples of hydrogen mass. The discovery of the neutron by J. Chadwick in 1932 solved this problem, and, for a short period of time, matter was believed to consist of only the three particles presented so far (2).

In 1900, M. Planck had already proposed the quantization of energy of the electromagnetic radiation emitted by a blackbody when he was trying to solve the ultraviolet catastrophe predicted by statistical mechanics applied to electromagnetic fields. He corrected that the energy emitted came in little packages of energy proportional to the frequency. A. Einstein adapted his results when explaining the photoelectric effect in 1905. Experiments led by R. Millikan in 1916 and, more notoriously by A. Compton in 1923 proved right Einstein's idea. Light exhibited wave properties such as interference or diffraction in macroscopic experiments, but Compton proved that at the microscopic level, it behaved as a particle - the photon, a name suggested by G. Lewis in 1926. Compton did it by finding that the wavelength of a light scattered from an electron is shifted. So there it was, Electrodynamics worked only as an approximation when huge numbers of photons are exchanged in processes given by Coulomb's law, while other processes involving individual photons call for quantization.

The question of how protons and neutrons are held together in the nucleus of an atom was also puzzling. The solution was duly termed "Strong Force", a short-range force with magnitude reaching a distance about 10^{-13}cm, roughly the size of the nucleus itself. In 1934, Yukawa proposed a theory for the strong force field, in which the quantum of the proton-neutron interaction was a meson particle. In 1947, after conflicting experiments, C. Powell and co-workers showed through photographic emulsions of cosmic rays that Yukawa's particle was the pion π, which decays into the muon μ and a neutrino, and the muon decays into an electron plus two neutrinos subsequently, having nothing to do with nuclear interactions (40).

$$(i\gamma^{\mu}\partial_{\mu} - m)\psi = 0 \tag{2.1}$$

It is worth introducing here how antiparticles fit into the whole picture. The brilliant P. Dirac combined SR with QM in 1927 deriving an equation (Eq. 2.1) for a free electron with relativistic energy, where i is the imaginary unit, γ^μ are a set of four matrices where $\mu = 0,1,2,3$ corresponds to spacetime coordinates and contains information about the spin and angular momentum of particles, ∂_μ are spacetime derivatives, m is the electron rest mass and ψ is the wave function.

Nonetheless, to account for the negative energy states produced by his equation when using the relativistic relation $E = \pm\sqrt{(pc)^2 + (mc^2)^2}$, he proposed the existence of a "sea" of negative energy, full of electrons. Whenever an electron is bounced off the sea when offered sufficient energy, it leaves a positive "hole" due to the Pauli exclusion principle [*], measured as a positively charged particle with positive energy. This particle, of course, should be identical to the electron, except for the positive energy. That new particle was soon to be discovered in 1931 by C. Anderson when analyzing cosmic rays tracks in a cloud chamber surrounded by a magnetic field. From the curvature of the track caused by the field, he was able to conclude that the particle was positively charged. Later, a more plausible formulation for this "sea" and its "holes" was given by Feynman and Stuckelberg, where the positive energy states are formed by positrons. One of the consequences of the Dirac equation was that for every particle produced in the universe, there had to be its corresponding charge counterpart, showing a natural symmetry between matter and antimatter. The caveat is that no such symmetry is observed within the observable universe and a preponderance of matter over antimatter is a problem that neutrinos may have a role in, and an experiment such as DUNE will try to address that.

2.2 Neutrinos

In 1933, E. Fermi developed a theory of beta decay in which the suggested neutrino particle by W. Pauli was included. Pauli originally proposed the neutron before Chadwick's discovery in 1932 to rescue conservation of energy in beta decays, which consist in a neutron decaying into a proton plus an electron plus a neutrino ($n \rightarrow p + e^- + \nu$).

[*]Formulated by Austrian physicist Wolfgang Pauli in 1925, the exclusion principle states that two or more identical fermions with half-integer spins cannot simultaneously occupy the same quantum state within a quantum system.

The experiments done prior to Chadwick's discovery, have shown that the electron energy in a beta decay would vary within a maximum value, suggesting that another particle was involved (Pauli's neutron). Fermi re-baptized Pauli's particle to neutrino due to its almost vanishingly small mass, according to the variations in electron energy. In the aforementioned Yukawa's pion decay, a neutrino was anticipated in the process due to the same reason, i.e. the energy of the muon varies, as well as in the secondary muon decay, in which the electron varied again, unless two more neutrinos are accounted for. But this was based only on theoretical grounds and the neutrino only served as reason for not abandoning energy conservation, one would say. Neutrinos rarely interact with matter, therefore an intense source and a big detector are necessary for studying them. In 1954, Cowan and Reines provided experimental proof of the neutrino's existence looking for the inverse beta decay $\overline{\nu} + p^+ \rightarrow n + e^+$ at a nuclear reactor, in which an antineutrino interacts with a proton, turning it into a neutron and positron.

At this point, one important question arises: how to distinguish the neutrino from it's antiparticle? Since the reaction $\nu + n \rightarrow p^+ + e^-$ must occur, Davis and Hermer in 1957 studied it with antineutrinos $\overline{\nu} + n \rightarrow p^+ + e^-$ and they found out that this process doesn't happen. The distinction between the neutrino and the antineutrino at that point was understood in the framework of a theory proposed 4 years earlier by Konopinski and Mahmud in which a lepton number L is assigned to these particles in reactions. The electrons, muons, and neutrinos are labeled $L = +1$, while $L = -1$ is reserved for their antiparticles. According to the theory, this number should be conserved, i.e., it should have the same value before and after the reactions. Thus, neutrinos and antineutrinos are distinct in Lepton number. Later, this Lepton number was split into a muon number L_μ and an electron number L_e to account for some "forbidden" decays, which also suggested that there are other kinds of neutrinos, each of which is associated with an electron and a muon, and later, the tau particle.

Once again, Davis in the 1960's, aimed to detect neutrinos from nuclear fusion reactions powering the Sun. He used a 100,000-gallon tank of tetrachloroethylene. Solar neutrinos would occasionally interact with chlorine-37 nuclei, converting them to radioactive argon-37 which could be chemically extracted and counted. Based on solar models, Davis expected to detect about 7 neutrino events per day in his tank. But over decades of running the experiment, he consistently found only about 1/3 as many neutrinos as

expected, around 2-3 events per day. This large discrepancy between the predicted and detected solar neutrino flux became known as the "solar neutrino problem". It was unclear if this meant solar models were wrong, neutrinos behaved unexpectedly, or detector results were incorrect. Solving this problem required further experiments with different detection methods to prove Davis's results were solid.

The ultimate solution was the neutrino flavor oscillation, with many solar neutrinos changing flavor in transit to Earth as their interaction with matter produces a difference in mass (9). This was showed in Sudbury Neutrino Observatory experiment in 2002 (41). Ultimately, neutrino oscillation plus the called Mikheyev-Smirnov-Wolfenstein (MSW) (10, 11) effect and a large mixing angle (LMA - the mixing between the mass and flavor of neutrinos) became the standard solution to the solar neutrino problem.

In 1998, the Super-Kamiokande experiment in Japan discovered oscillations in atmospheric neutrinos created by cosmic rays (13). This proved neutrinos must have mass since oscillations require mass differences between neutrino types. It also revealed three neutrino flavors mixed in transit. Further experiments found that mass differences driving these oscillations are small, and yet the mixing between flavors is large (14). This oscillation plays a role in theories explaining why the observable universe contains practically only matter, with almost no antimatter observed from the Big Bang (15).

There are two key parameters in the MSW-LMA solution mentioned: First, the mixing angles θ_{12}, θ_{13}, θ_{23} describe the probability of one type of neutrino (e.g., electron neutrino) transforming into another type (e.g., muon or tau neutrino) as they travel through matter. The larger the mixing angle, the higher the probability of oscillation between different neutrino flavors. Second, the Δm^2_{21}, Δm^2_{23}, Δm^2_{31} parameters refer to the mass differences between the neutrino mass eigenstates. This parameter influences the frequency and amplitude of neutrino oscillations.

$$i\frac{d}{dx}\nu_i = H\nu_i \tag{2.2}$$

The survival probability of solar electron neutrinos can be obtained by solving the Schrödinger equation (2.2), which describes the evolution of neutrino flavors across a propagation distance x. The three eigenstates corresponding to the neutrino flavors ν_1,ν_2, ν_3 form a basis in Hilbert space which can diagonalize the Hamiltonian H for a free neutrino, leading to a superposition of mass eigenstates, shown in equation 2.3. $U_{\alpha\beta}$ is

a "mixing matrix", whereas β is an amplitude of the mass eigenstate in terms of α, the flavor eigenstate. Thus, the unitary matrix $U_{\alpha\beta}$ parameterizes the unitary transformation between the two bases, and the probability of measuring a neutrino α with mass m_β is $|U_{\alpha\beta}|^2$.

$$\nu_\alpha = U_{\alpha\beta}\nu_\beta \tag{2.3}$$

The standard parametrization in terms of the mixing angles of the $U_{\alpha\beta}$ matrix is the so called Pontecorvo–Maki–Nakagawa–Sakata (PMNS) matrix and it can be used to solve the Schrödinger equation using approximations such as the adiabatic approximation, which assumes that the matter density varies smoothly so that if a neutrino is in an mass eigenstate, then in the course of propagation it won't oscillate into another matter eigenstate. The neutrino oscillation model is today defined in terms of these ideas and the mass squared differences between the neutrino flavours is what experiments try to address.

Nevertheless, many open questions remain, including whether neutrinos are Majorana particles[†] the problem of the neutrino mass hierarchy and the exact details of the neutrino mixing parameters exposed.

2.3 The Deep Underground Neutrino Experiment (DUNE)

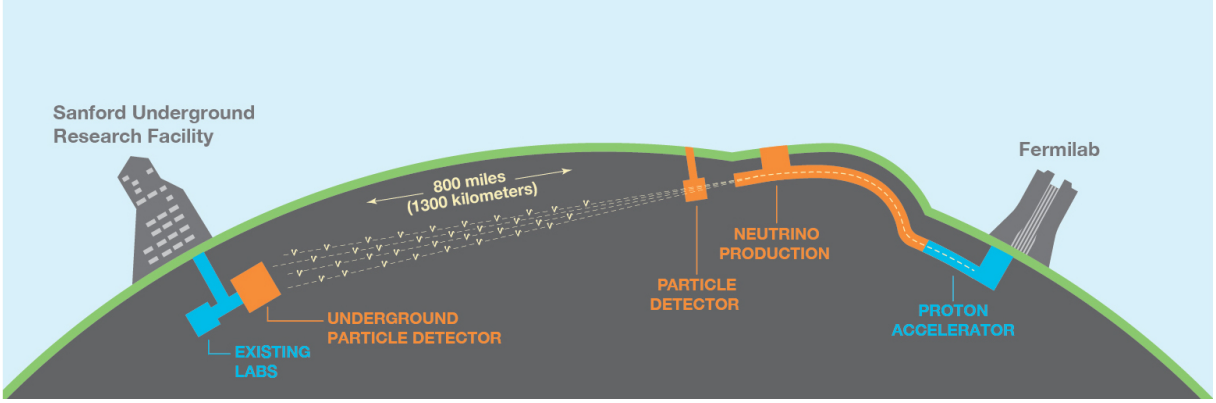
The DUNE (Figure 2) will be a neutrino and nucleon decay detector intended to address fundamental questions about the nature of elementary particles and their role in the universe. Specifically, the goal of DUNE is discovering matter-antimatter asymmetries in neutrino flavor mixing, precisely measure neutrino oscillation parameters, including the mass ordering Δm^2_{31} and the mixing angle θ_{23}, the observation of proton decay and possibly the electron neutrino ν_e flux from supernovae core-collapse (18).

Hosted by the Fermi laboratory (Fermilab) in the USA, DUNE will consist of a far detector (FD) to be located about 1.5 km underground at the Sanford Underground Research Facility (SURF) in South Dakota, USA, at a distance of 1300 km from Fermilab

[†]Majorana particles are in theory their own antiparticles, and were predicted in 1937 by Italian physicist Ettore Majorana.

and a near detector (ND) to be located at Fermilab in Illinois.

Figure 2 – Graphical representation of the DUNE setup.



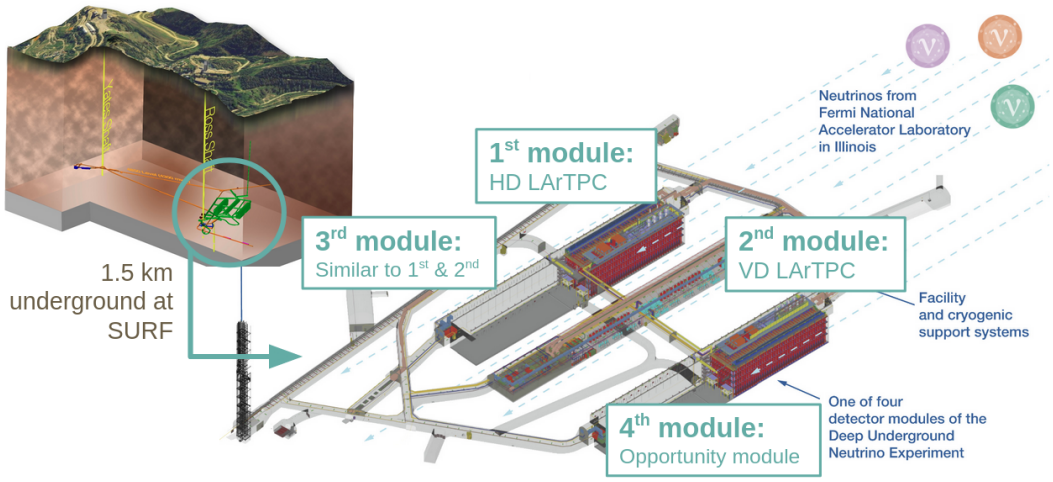
Source: (18)

The far detector will be a large liquid argon time-projection chamber (LArTPC) with four detector modules, each contained in a cryostat holding 17.5 kt of liquid argon (LAr) with 10kt of mass each. This LAr technology will make it possible to reconstruct neutrino interactions with image-like precision and unprecedented resolution due to its tracking and calorimetry performance (18). The far detector contains the Arapucas light sensors that are of interest in this work.

At Fermilab, an intense neutrino beam will be generated targeting the far detector. Closer to Fermilab, at 575m from the beam generator, the near detector will be used to measure the neutrino beam flux and flavor composition. The experiment involves sending beams of neutrinos and antineutrinos. Comparing the neutrino energy at the ND and FD allows the differentiation of effects that controls the beam energy spectrum and reduce uncertainties, necessary for understanding charge parity (CP) violation[‡], for instance, one of DUNE's goals. By studying the behavior of neutrinos and antineutrinos in the FD in seeing how they oscillate and interact, one can understand if there are differences in their behavior, leading to insights concerning CP violation (12). The ND will measure neutrino-argon interactions, which can also reduce the uncertainties associated with the modeling of these interactions (18). The infrastructure for these detectors will be maintained under responsibility by the Long-Baseline Neutrino Facility (LBNF) at the Illinois and South Dakota Fermilab units.

[‡]CP symmetry states that the laws of physics must be the same if a particle is exchanged for its antiparticle while their spatial coordinates are reversed.

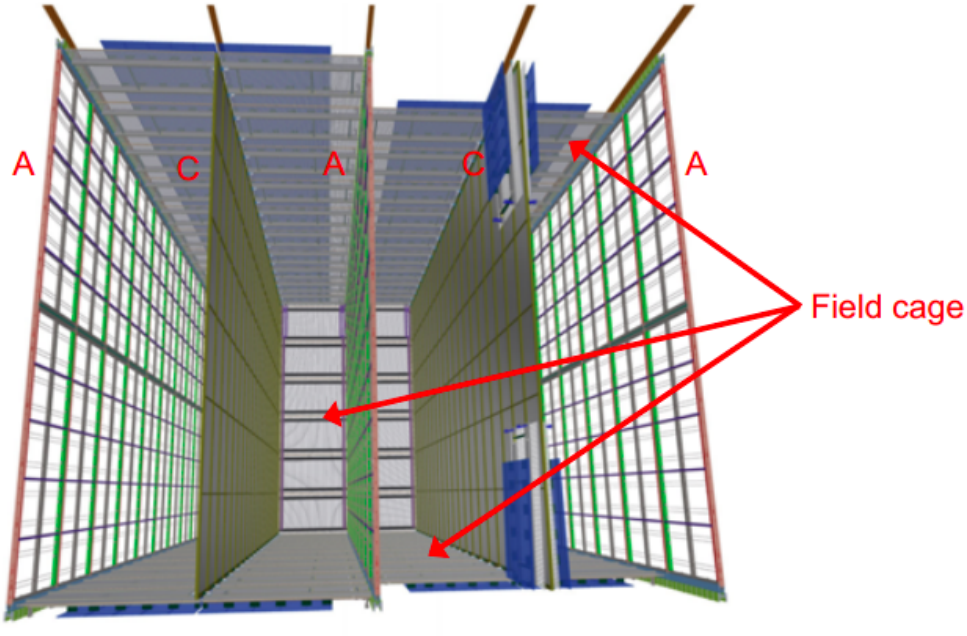
Figure 3 – Representation of DUNE's FD detectors at Sanford (SURF) facilities.



Source: (29)

There are two LArTPC's technologies being constructed for DUNE. The first, using is a Single Phase detection and the second using a Dual Phase system. Out of the four FD detector modules (Figure 3), the first FD module will harness the Horizontal Drift (HD) technology in the Single Phase, while the second module the Vertical Drift (VD) in the Dual Phase and in 3rd and 4th modules of the FD the technology are still being planned. HD and VD are the way ionization charges from neutrino reactions drift in the LAr chambers and are deposited in the wires. The LAr emits scintillation light at 127nm wavelength once the event occurs and the photons can then be captured by the Arapucas. This give the starting point for the drifting, which can in turn lead to the software reconstruction of the entire process.

Figure 4 – Configuration of one DUNE single phase module containing the anode and cathode plane assemblies.



Source: (18)

Charged particles passing through the chamber ionize the LAr, and the electrons drift in an electric field to the "walls" (Figure 4). These walls are the anode and cathode plane assemblies, APA and CPA, respectively. The single-phase detector has 150 APAs and each of them is two-sided with layers of wires forming a grid. The Arapucas are arranged together with the wire planes of the APAs and there are 10 of them for each APA. The Arapucas are core components in DUNE detectors.

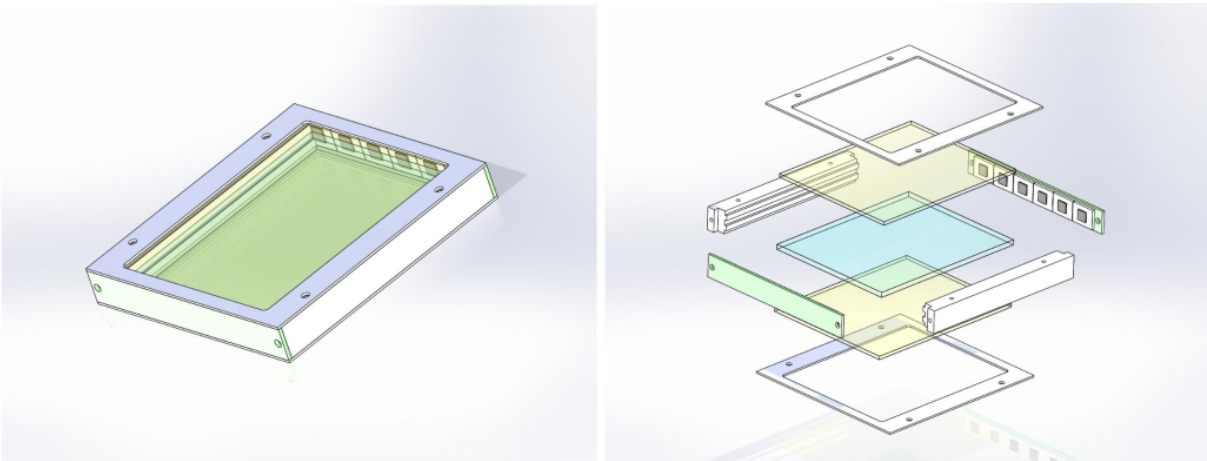
The DUNE collaboration was formed in 2015 and includes scientists and engineers around the globe, representing the union of a neutrino physics research community. This made possible the development towards a long-baseline neutrino experiment over the last decade, under investment provided the U.S. Department of Energy and Fermilab to support an expansion of the infrastructure in South Dakota, and to create a megawatt neutrino-beam facility at Fermilab by 2026. By 2030, the plan is to enable accelerators to increase neutrino beam power.

2.3.1 The Arapuca Device

The X-Arapuca, a version of the Arapuca device, has been proposed as the photon detection system for the LArTPC's far detector of the DUNE neutrino experiment. Introduced in 2016, the Arapucas aimed to detect scintillation light in liquid noble gas detectors like liquid argon.

It uses a dichroic filter as an optical window to trap scintillation photons inside a box with highly reflective surfaces. Figure 5 shows a representation of the X-Arapuca. The dichroic filter transmits light below a cutoff wavelength but reflects light above the cutoff. The filter is coated on the outside with a wavelength shifter called para-Terphenyl (PTP) that shifts the 127 nm argon scintillation light to around 350 nm so it can pass through the filter. Inside the box there is another wavelength shifter, tetra-phenyl butadiene (TPB), that further shifts the light to around 430 nm. This light gets reflected by the dichroic filter and box surfaces 6.

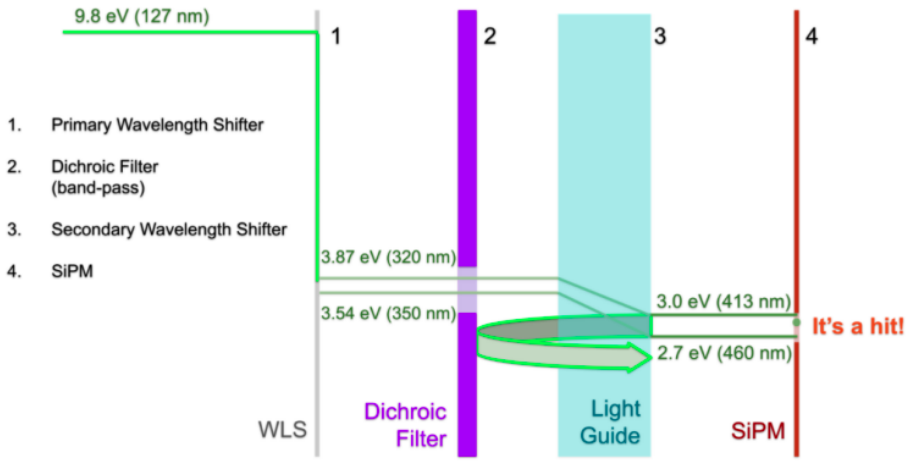
Figure 5 – Representation of the X-Arapuca device.



Yellow rectangular plates are the dichroic filters coated with PTP and the blue plate is the wavelength shifter. Source: (6)

By employing a combination of wavelength shifters and dichroic filters (figure 6), the photons are trapped until they hit a silicon photomultiplier (SiPM) sensor on the inner surface within an enhanced probability. This photon trap allows reaching 1% detection efficiency with only a small fraction (0.1%) of the inner surface covered by the SiPM sensor. It can be used in large neutrino and dark matter detectors over wide areas without needing full coverage with expensive photosensors (4).

Figure 6 – The X-Arapuca scheme showing the two stages of photon downshifting with a dichroic filter and a WLS coupled to SiPMs.



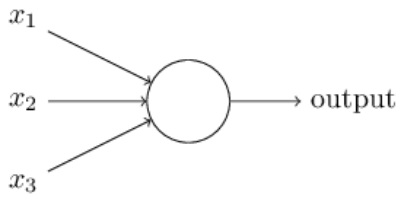
The PTP coating of the device entrance window downshifts photons from 128 nm to 350 nm, re-emitting about 50% of them inside the trap that is full of liquid argon. The photons absorbed by the WLS are again re-emitted at 440nm and are driven to the SiPMs. The dichroic filter has a cutoff at 400 nm and the photons bounce back and forth until they are finally either detected by SiPMs or lost. Source: (7, 8)

In a recent long-term testing report using a 2-window X-Arapuca, the device was installed in a cryostat filled with liquid argon. An Americium-241 alpha source provided scintillation light signals. The output was read out continuously for 10 days to monitor stability. The light yield showed <1% variation after initial stabilization, indicating a stable system (5). The absolute photon detection efficiency was estimated to be 2.5%, by comparing detected photoelectrons to the expected number of photons reaching the surface from the source. This is a key requirement for future massive neutrino and dark matter experiments such as DUNE. For collecting the data set for this work, a simulation of the X-Arapuca was used. More on this in the sections to come.

2.4 Neural Networks

Neural networks emerged as a tool in the field of artificial intelligence and are a subset of machine learning area. Machine learning algorithms can be categorized into different types, such as supervised learning, unsupervised learning, and reinforcement learning. They are designed to learn patterns and relationships from training datasets, harnessing similar mechanisms from the structure and functionality of the human nervous

Figure 7 – Example of an artificial neuron model by McCulloch and Pitts.



The input signals should meet a condition of activation, generating a binary output. It works as a logical gate, following a boolean logic. Source: The author

system, along with theories on neuron excitement and communication.

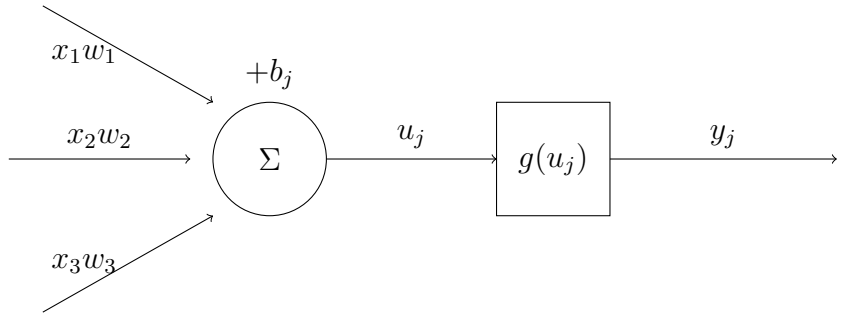
The foundation for the biological neuron doctrine is attributed to Santiago Ramón y Cajal's work in the late 19th and early 20th centuries (31). Cajal's observations and illustrations of neurons under the microscope helped establish the understanding that the nervous system is composed of individual cells called neurons, which communicate through specialized connections known as synapses. In the mid-20th century, researchers such as Alan Hodgkin and Andrew Huxley made significant discoveries about the electrical properties of neurons. They elucidated the mechanism of action potentials, which are brief electrical pulses that allow neurons to transmit signals over long distances once a certain electrical threshold is met (32). Additionally, scientists such as Otto Loewi and Bernard Katz contributed to the understanding of synaptic transmission, the process by which neurotransmitters are released from one neuron and received by another (33).

In the field of neuropsychology, Donald Hebb's work played a crucial role in bridging the gap between neurobiology and learning algorithms. He proposed Hebb's rule, which suggests that when two neurons persistently activate each other, the connection between them is strengthened, famously stating that "cells that fire together, wire together". Hebb's rule laid the foundation for understanding synaptic plasticity, the ability of connections between neurons to change in strength based on experience and learning (30). These foundational discoveries in neuroscience and neuropsychology serve as the backbone of modern neural networks.

The phrase "machine learning" was coined by Arthur Samuel while working on IBM in 1952 developing computer programs for checkers playing, and it's meaning was defined a decade ago by Stanford University as "the science of getting computers to act without being explicitly programmed" (28).

W. McCulloch and W. Pitts presented the first computational neuron model (19),

Figure 8 – Example of Rosenblatt's perceptron model.



A neuron is activated when the sum of inputs, weights, and biases meet a certain threshold, generating an output y_j. Source: The author

in which the neuron activity is a binary process given by the synapses that activates the neuron within a threshold (Figure 7). Their model was improved by F. Rosenblatt at the Cornell Aeronautical Laboratory, introducing the perceptron (Figure **??**), a single-layer neural network capable of learning and making binary classifications (26). It was originally designed for the IBM 704 computer and installed in a custom-built machine called the Mark 1 Perceptron, which had been constructed for image and pattern recognition tasks. While the initial perceptron model had limitations in terms of its capability to handle complex tasks, being successful only in recognizing simple patterns, subsequent research led to the progress of neural networks.

A proper neural network is a complex network of interconnected artificial neurons organized in layers, as exemplified in Figure 9. This characterizes the Multi-Layer Perceptrons (MLP's) and its observation in the 1960s, that incorporating multiple layers of neurons in Rosenblatt's perceptron model significantly increased processing power was a pivotal moment. The so-called Deep Learning was born, being characterized as neural networks composed of many layers. This breakthrough paved the way for the development of various neural network architectures, leading to a continuously expanding variety of them. One notable outcome of incorporating multiple layers is the advent of feedforward neural networks and the introduction of backpropagation techniques for training (27).

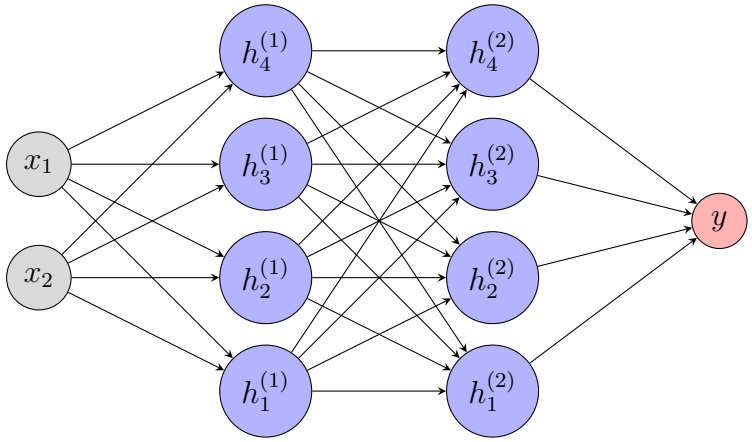
Computational neural network models have gained attention for their ability to learn patterns from data and perform complex tasks, demonstrating remarkable success in various domains, such as image classification (22), natural language processing (23), speech recognition (24) and sequential data processing (25). Besides, there have been significant advancements in the past decade in leveraging graphics processing units (GPUs)

to improve computer power destined for deep learning. Although GPUs were originally designed for rendering graphics, they emerged as powerful accelerators for parallel computing - that is, running many calculations at the same time, being extremely useful for training neural networks. With that development, deep learning libraries and frameworks of programming languages such as TensorFlow (16) and PyTorch (36) became available, which provide high-level abstractions and optimization techniques when applying GPU into neural networks.

As exemplified in Figure **??**, each artificial neuron in the network receives data x_i as its input, numbers for example, applies a weighted sum operation on it (Equation 2.4), and passes the result through an activation function $g(u_j)$ (Equation 2.5) to produce an output y_j.

$$u_j(x_i) = \sum_{i}^{n} w_{ij} x_i + b_j \tag{2.4}$$

Figure 9 – Example of an artificial multi-layer neural network with an input (grey neurons), an output (red neuron) and 2 hidden layers (blue neurons).



Each arrow connecting the neurons may be called a "synapse" and has a weight w_{ij} associated with it. Source: The author

$$y_j = g[u_j(x_i)] = \frac{1}{1 + e^{-u_j}}. \tag{2.5}$$

The weights, w_{ij}, determine the strength and influence of each neuron's contribution to the overall computation. They are multiplied with the inputs of a neuron to determine the importance or contribution of each input towards the neuron's output. The activation function, on the other hand, is responsible for introducing nonlinearity to the

output. It acts as a filter or threshold, determining which signals are relevant or important for the subsequent computations. The b in Equation 2.4 stands for *bias*, a constant value added to the weighted sum of inputs before applying the activation function and it's not affecting the weights themselves overall. It has a similar role as an initial value for solving a differential equation, providing an offset or a reference point for the activation function, allowing the neural network to capture patterns that may not pass through the origin. When it comes to the activation function, the choice of function depends on the specific problem at hand. Different activation functions introduce different nonlinearities. A commonly used activation function is the sigmoid (Equation 2.5), which squashes the input into a range between 0 and 1. It is useful in scenarios where the output needs to be interpreted as a probability or when dealing with binary classification problems. Other similar activations are the ReLU, that sets all negative values to zero, effectively promoting sparse activations within the network, the hyperbolic tangent ($tanh$) function, which squashes the input between -1 and 1, and the softmax function, used in multiclass classification problems to produce a probability distribution over multiple classes.

There are also several neural networks models available. The one described above emerged in the context of the already mentioned MLP model, first introduced by (27). The MLP consists of an input layer, one or more hidden layers, and an output layer, characterizing a feed-forward architecture. The hidden layers are responsible for extracting meaningful features from the input data, while the output layer produces the final prediction. Particularly for this work, the process involved in training a neural network is called supervised learning. In this process, the network is presented with a labeled dataset, where both the inputs and their corresponding outputs are known. The network adjusts its weights and biases through an iterative optimization algorithm to minimize the difference between its predicted outputs and the true outputs. This is commonly done using the backpropagation mentioned earlier.

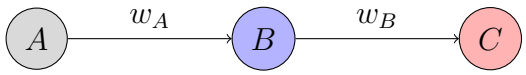
The backpropagation technique propagates the error signal backward through the network, enabling the adjustment of weights in a way that reduces the overall error. This error is commonly referred to as the "loss" or "cost". The loss represents the discrepancy between the predicted outputs of the neural network and the actual target values. The goal of the backpropagation algorithm is to calculate the gradients of the loss concerning the weights and biases in the network. The loss is measured using known statistical

metrics, such as Mean Squared Error (MSE) or Cross-Entropy Loss.

2.4.1 Backpropagation Algorithms

Backpropagation is one of a more general technique called "reverse mode differentiation" to compute derivatives of functions represented in some directed graph form, but for this work, it is enough to explain only backpropagation itself. It works as follows: Considering the MSE loss function, let's say that for a certain supervised training example of one neuron in each layer (Figure 10), the desired output is y_0.

Figure 10 – Simple neural network with one neuron in each layer.



Source: The author

The loss is then

$$l_0 = (C - y_0)^2 \tag{2.6}$$

This last neuron's activation is given by weight, bias, and the previous neuron's activation, all done by a function

$$C = f(w_C B + b_C) = f(u) \tag{2.7}$$

The same goes for the activation of B, of course.

This activation function can be of the sigmoid form 2.5, or the rectified linear activation function (ReLU), for example.

If the weights and biases change a bit, it influences u, which in turn influences the last neuron's activation, altering the loss a bit. The changes are monitored by deriving the loss function with respect to the weight and using the chain rule,

$$\frac{\partial l_0}{\partial w_C} = \frac{\partial u}{\partial w_C}\frac{\partial C}{\partial u}\frac{\partial l_0}{\partial C}. \tag{2.8}$$

Performing the derivatives in 2.6 and 2.7,

$$\frac{\partial l_0}{\partial w_C} = Bf'(u)2(C - y_0) \tag{2.9}$$

The same is done for the bias,

$$\frac{\partial l_0}{\partial b_C} = \frac{\partial u}{\partial b_C}\frac{\partial C}{\partial u}\frac{\partial l_0}{\partial C} = f'(u)2(C - y_0). \tag{2.10}$$

The idea of propagating backward into the network comes in hand when the variation of the loss function is analyzed in terms of the previous layer neuron's activation

$$\frac{\partial l_0}{\partial B} = \frac{\partial u}{\partial B}\frac{\partial C}{\partial u}\frac{\partial l_0}{\partial C} = w_C f'(u)2(C - y_0). \tag{2.11}$$

One might think that if the problem evokes a more complex neural network similar to that in Figure 9, the equations above might as well get complex. But in fact the same process described applies, all that would need to be done is introduce new indices to keep track of all the additional neurons. Let B_j be the neurons in the hidden layer and C_k neurons in the output layer. Equation 2.6 becomes

$$l_0 = \sum_{k=0}^{n-1}(C_k - y_k)^2. \tag{2.12}$$

The weights liking the hidden and output layer can then be labeled w_{kj}^C and u in Equation 2.7 becomes

$$f(u_k^C) = f(w_{kj}^C B_j + b_k^C). \tag{2.13}$$

Equation 2.8 stays pretty much the same as before

$$\frac{\partial l_0}{\partial w_{kj}^C} = \frac{\partial u_k^C}{\partial w_{kj}^C}\frac{\partial C_k}{\partial u_k^C}\frac{\partial l_0}{\partial C_k}. \tag{2.14}$$

A rather subtle difference comes in when the differential of the loss is analyzed in terms of the activation of the previous layer's neurons (Equation 2.11)

$$\frac{\partial l_0}{\partial B_j} = \sum_{k=0}^{n-1}\frac{\partial u_k^C}{\partial B_j}\frac{\partial C_k}{\partial u_k^C}\frac{\partial l_0}{\partial C_k}. \tag{2.15}$$

That means the j-th neuron in the previous layer influences the loss function through different k-th synapses, requiring their contributions to be summed up. This

process can then be repeated in reverse for the weights and biases feeding into the j-th layer.

These equations form the foundation of backpropagation, providing the rates of change necessary for the network to learn. Each component of the gradient vector of the loss function contributes to its minimization through an iterative process. Today, backpropagation is being used to train deep neural networks in general.

Only the MLP architecture is described here up to now. Moreover, other architectures revolutionized machine learning. Here's a comment on the most popular ones:

- Convolutional Neural Networks (CNNs): CNNs are widely used for image processing tasks due to their ability to capture spatial relationships in data. They employ convolutional layers that apply filters to input data, enabling automatic feature extraction. They are used in image classification, object detection, and image segmentation. (34)

- Recurrent Neural Networks (RNNs): RNNs are designed to handle sequential or time-series data, making them suitable for tasks involving temporal dependencies. They have recurrent connections that allow information to persist across time steps, enabling the modeling of sequences. RNNs have been applied in speech recognition, natural language processing, and time series prediction. (20)

- Long Short-Term Memory (LSTM) Networks: LSTMs are a specialized type of RNN that addresses the vanishing gradient problem by introducing memory cells. These cells maintain memory over long sequences, allowing for better learning and capturing long-term dependencies. LSTMs have been used in machine translation, sentiment analysis, and speech recognition tasks. (25)

- Transformer Models: Transformer models revolutionized natural language processing tasks, particularly in machine translation, with the introduction of the Attention mechanism. They rely on self-attention layers to capture relationships between different words (tokens) in a sentence, eliminating the need for recurrent connections. Famous transformer models are GPT and BERT. (23)

2.4.2 Optimization Algorithms

In the process of minimizing the loss function in training a neural network, there are several techniques available. Optimization algorithms, for example, play a role in training neural networks by updating the model's weights and biases to minimize the loss. One common and fundamental algorithm is gradient descent, which calculates the gradient of the loss function concerning the model parameters and adjusts the parameters in the direction of steepest descent (35). One can establish a certain step in the learning process determining the step size of each update, impacting the convergence speed and stability of the algorithm. Such a step in the learning process is called learning rate, it is one of the parameters that compose most of the architectures used and is included in the gradient descent algorithm, which is usually written in the form of Equation 2.16

$$\theta_{i+1} = \theta_i - \alpha \frac{\partial l(\theta_n)}{\partial \theta_i}. \tag{2.16}$$

The parameters used in the learning process by the network are the weights and biases implicit in the variable θ, and the α correspond to the learning rate. The learning rate constitutes one of the so-called hyperparameters of the neural network, which are parameters set before the training starts, like the number of layers in the network, the choice of activation functions, the batch size, etc. These hyperparameters are not involved in the training and are rather arbitrary choices made by the researcher in finding an optimal combination that satisfies his needs. This process constitutes what is called hyperparameter tuning, and their methods for doing so are brought by packages in programming languages used to train neural nets. More on this in section 4.

There are variants of gradient descent like batch gradient descent, stochastic gradient descent (SGD), and mini-batch gradient descent. Also, one can enhance gradient descent with adaptative optimization such as Adam, RMSprop, and Adagrad algorithms - all of these being specific to each neural network trained.

3 OBJECTIVE

The general and primary objective of this study is to utilize a Machine Learning method to reproduce probabilities of scintillation light absorption from a simulation of an X-Arapuca device - a version of the Arapuca family - using the ArapucaSim software. To achieve this, it is employed a supervised training approach using neural networks trained on data generated by Geant4 libraries using Monte Carlo simulation methods, which are based on stochastic processes and are utilized for generating data sets for complex problems in Particle Physics. However, for deterministic problems, neural networks have demonstrated superior efficiency compared to these methods in terms of time and computer power. Therefore, by leveraging neural networks in trying to mimic the simulation process, an optimization in gathering probabilities of light absorption by the X-Arapucas using a state-of-the-art model is foreseen.

Thus, the specific objectives of this study are twofold:

a) Customized data sets derived from the simulation of the absorption probability response of the X-Arapucas are created using the ArapucaSim software built over the Geant4 framework. These data sets will be employed to train neural networks with the goal of producing probabilities of photon absorption within the device with the lowest possible error.

b) Parameters during the neural network training process are explored, including number of neurons, activation functions, optimizers, correction factors, among others. By conducting comparisons and employing statistical metrics, the neural network performance will be test in trying to get the most accurate results, while applying the minor computational time and cost.

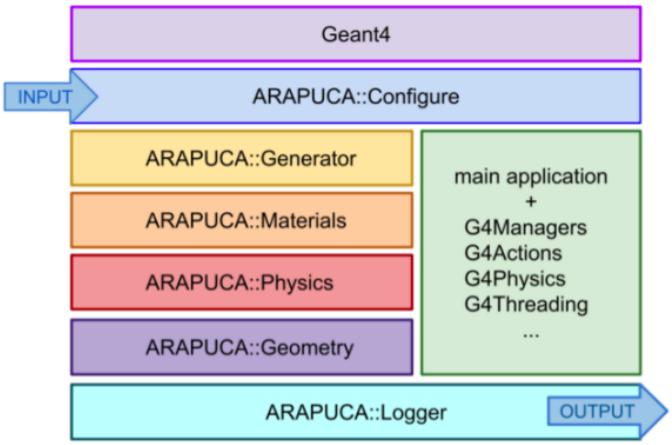
4 RESEARCH METHODOLOGY

Machine learning techniques can be divided into two methods: unsupervised and supervised learning. This work is based on models of the latter kind, characterized by the fact that inputs and outputs are known variables organized in a table or text file. Two fundamental steps are employed in dealing with this type of datset: a pre-processing of the collected data and the application of artificial neural networks as a regression problem where the outputs are continuous values, both of which are described in the following sections. Several neural network models were tested using models available online and one of them produced satisfactory results using the TensorFlow framework (16).

4.1 The data sets collection with ArapucaSim

The input data collected for training the NN's was generated using the ArapucaSim simulation, an open-source code built on top of Geant4, a very known toolkit used in High-Energy Physics to model the passage of particles through matter.

Figure 11 – The ArapucaSim software structure.

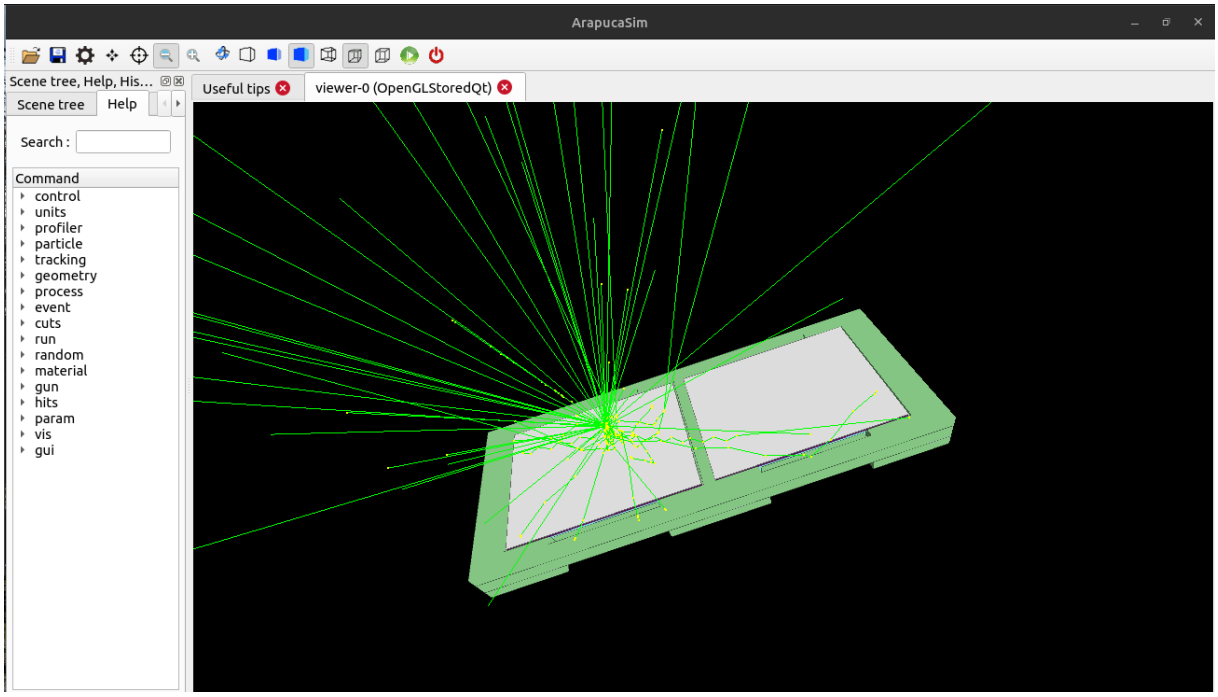


Source: (7)

The ArapucaSim software source code include managers for generating photons, defining the geometry of the simulation and its materials, together with the physics behind it reproducing the wavelength shifting, scintillation and dichroic filters, as shown in the

above diagram of Figure 11. Figure 12 shows an example of an interactive run of the ArapucaSim program, available at Github. In this simulation, a dual-cell X-Arapuca is used - two devices side by side.

Figure 12 – The ArapucaSim interactive mode allows for visualization of incoming photons at the surface of the simulated device.

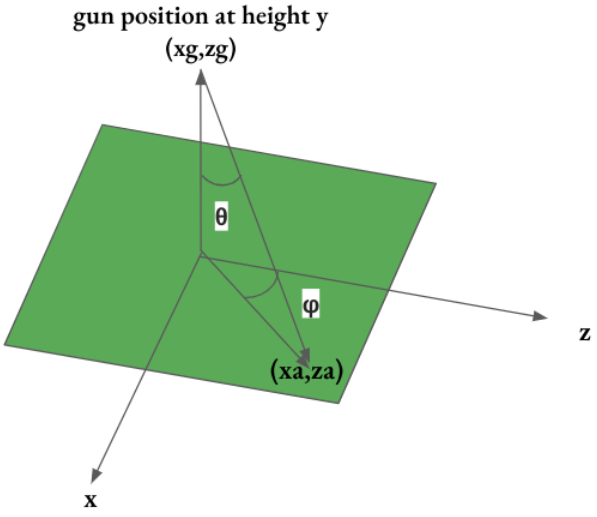


Green lines represent photons trajectory. Some are absorbed, as represented by the track left inside the device, while most of them are bounced back. Source: The author

The ArapucaSim can receive several commands from the user as long as they are previously defined in the source code. The necessary commands for a simulation one is trying to run are: the position in cartesian (x_g,y_g,z_g) or polar coordinates (θ,ϕ) from where the photons were generated in the "gun"; the position in cartesian coordinates of where photons are arriving at the surface of the device (x_a,z_a), and the number n of photons being targeted. Although not required for the purposes of this project, the energy of the incoming photons can be changed as well. All this parameters are flags in the command line when executing the program or defined in a separate script. A Python code, detailed in Annex I, was written to run the program with iterations over ranges of these coordinate values, mapping the device surface. Figure 13 shows a diagram of the relationship between the coordinates. The dual-cell X-Arapuca dimensions in this simulations are 24cm by 9.6cm, in $x_a : [-12cm,12cm]$ and $z_a : [-4.8cm,4.8cm]$. After

running the script, the output is a csv file containing the number of photons captured for each combination of (x_a,z_a,x_g,z_g) or (x_a,z_a,θ,ϕ), which are labeled as a percentage value $e = n_{absorbed}/n_{targeted}$.

Figure 13 – The python script running the ArapucaSim iterates over pairs of values (xa,za) varying (xg,zg,θ,ϕ).



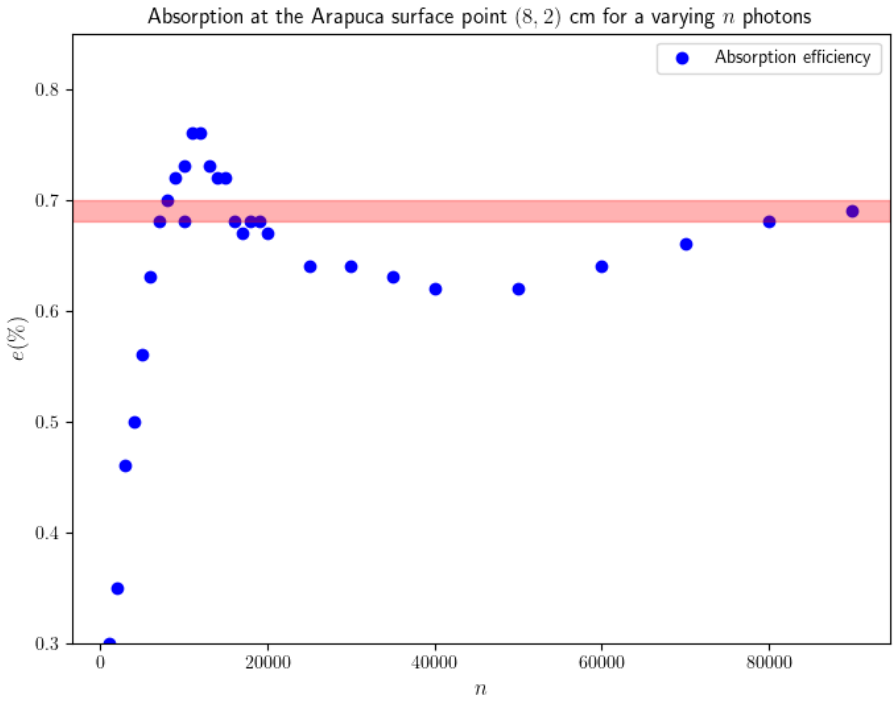
For each of those values, a number of photons absorbed is associated. The photon generator is fixed at $y = 10cm$ and the Arapuca is fixed at $y = 0$. Source: The author

Two data sets were collected: the first was efficiencies corresponding to normal incidence photons, i.e. where $x_a = x_g$ and $z_a = z_g$. The second, iterations over angles θ, and ϕ were included. The resulting data was used to be fed to the neural network is referred to as a tabular regression data set, in which the goal is to predict a continuous numerical value based on one or more input features. The number of incidence photons targeted towards the Arapuca plane was chosen to be 10.000 (Figure 14).

ArapucaSim uses a stochastic approach to simulate light-matter interaction. Therefore, in principle, an infinite number of photons should be fired over each point of the Arapuca device to get an absorption probability, which is an out-of-hand task. However, looking at the convergence of the efficiency as a function of the number of photons, the latter being one that corresponds to the medium value would be a good choice. The curve in Figure 14 was obtained by choosing a single point at the Arapuca surface to be hit by photons. Any point x_a,z_a could be chosen that the absorption efficiency, although different for each point, as it will be seen in section 5.1, the number of photons converge by the same token.

Therefore, the inputs are coordinates and the outputs are absorption efficiencies, such that $e = e(x_a^i, z_a^j, \theta^k, \phi^l)$ (the upper indexes indicate the ranges of each variable in the Python iterated loop, that will be detailed in the next paragraphs).

Figure 14 – Efficiency of photon absorption as a function of the number of photons being hit at the Arapuca surface.



Source: The author

4.2 The Neural Network training

First and foremost, how to choose an appropriate NN model architecture for the task at hand? It surely depends on the goal of the problem, the type, complexity, and size of data. As mentioned previously, the goal here is to use a supervised model, since the output is a known continuous variable. The data set is a somewhat complex and noisy relationship between 4 coordinates and a percentage value. When it comes to the size of the data set, the option was to get as much points as possible from the simulation. Of course, the computational cost and time increases as the number of points are higher. For the normal incidence photons, a data set of roughly 25 thousand points was collected. As for the oblique incidence photons, approximately 75 thousand points were collected. A csv file "coordinates.csv" (python script in Annex 1) was created separately to contain

those points to be used by the ArapucaSim.

To train the neural network effectively, the first requirement is a preprocessing and appropriate classification of the data set. Preprocessing involves preparing the data to the learning process of the network, such as exemplified in Listing 1, where the data is re-scaled and split up. This crucial aspect of preprocessing is standardization, where the inputs and outputs are adjusted to be within a well-defined range, ensuring that they are in a format that the neural network can work with.

```
input_csv = pd.read_csv('output-arapucasim.csv') #opening the data set
gun_coord = input_csv[['xg','zg']].values
eff = input_csv['e'].values

scaler = MinMaxScaler() #normalization
gun_norm = scaler.fit_transform(gun_coord)

coord_train, coord_test, eff_train, eff_test = train_test_split(gun_norm
    , eff, test_size=0.2, random_state=42) #splitting the data into
    training/testing
```

Listing 1 – Example of a Python script to preprocess and classify the data to be trained. Inputs are adjusted to a range between 0 and 1 using the MinMaxScaler function from the Scikit Learn library. The last line splits up the input and output into data to be trained on and data to be tested against. Here x_g and z_g are the gun (the photon generator) coordinates and e is the efficiency of absorbed photons. Source: The author

In this study, the neural networks models employed are types of neural network called Multi-Layer Perceptron (MLP) models. The MLP belongs to the class of feed-forward artificial neural networks (FNN's), which are structured in layers composed of multiple neurons and activation functions, constituting a deep neural network. The advantages and reason for this choice is that they can accommodate large data sets, capture non-linear dependencies between variables in the data.

As explained in section 2.4, the connections between neurons are unidirectional, with inputs flowing from one layer to the next. This means that the outputs of neurons in one layer become the inputs for neurons in the subsequent layer. The activation level of each neuron is determined by its weight, which specifies the degree of influence it has on the network as a whole. By properly adjusting weights and biases, the network can

model the correlations among input and outputs in the training dataset.

The neural networks allows for statistical regularization techniques such as dropout and L2 regularization, which will be later described, in preventing overfitting. There are several parameters to be set to get the architecture that better regresses the dataset's correlations. Those parameters are referred to as hyperparameters. There are options to "tune" these hyperparameters to arrive at a more compact and simpler model, which will be later discussed.

Let's briefly describe the most important parameters of interest: The **Architecture** refers to the neural network's structure, including the number of hidden layers and neurons in each layer. **Activation**: Specifies the activation function ReLU (Rectified Linear Unit) used in the hidden layers. **Solver**: The optimization algorithm used for training the neural network, with 'Adam' being an adaptive optimization algorithm. **Batch Size**: Determines the number of data samples used in each forward and backward step during each epoch. **Regularization**: Controls the L2 regularization strength, which helps prevent overfitting by adding a correction to the loss function based on the weights. **Learning Rate**: Sets the initial learning rate, which determines the step size in updating model weights in each epoch. **Tolerance**: Defines the convergence criteria for an "early stopping", i.e. the training stops when the validation score does not improve by more than this value. **Validation Fraction**: Determines the proportion of the initial data reserved for testing at each epoch. **Batch Normalization**: A feature that normalizes the activation function in a layer to stabilize training. **Learning Rate Scheduling**: This function makes the learning rate dynamic during training, which can improve convergence. **Dropout Regularization**: A technique that randomly drops out neurons during training to prevent overfitting, making it more dynamic.

Table 1 – Hyperparameters and features of the Neural Network model using Keras framework.

Architecture	**Training Settings**	**Performance Metrics**	**Other Features**
4 Hidden Layers	Solver: Adam	MSE	Batch Normalization
512 neurons/layer	Activation: ReLU	MAE	Early Stopping
ReLU activation	Batch Size: 64		Learning Rate Scheduling
Reg. L2 = 0.001	Epochs: 200		Dropout Regularization
	Learning Rate: 0.0041		

Source: The author

The neural network model provided in Annex II is defined using the Sequential API from TensorFlow Keras. The model architecture was first based on examples provided by Tensorflow[§] and it's final version is summarized in Table 1.

Table 2 – Models variations performance for comparison, showing neural networks with different hidden layers (HD) and their correspondent number of neurons per layer.

Architecture	**Trainable Parameters**	**MSE**	**MAE**
3 HL, [192, 128, 512] Neurons	93889	0.08	0.09
3 HL, [96, 384, 32] Neurons	60609	0.07	0.09
2 HL, [1840, 32] Neurons	71889	0.10	0.12

Source: The author

Table 1 contains the most satisfactory parameters found so far for the models used. By "found" it is meant that the initial model was obtained from the simplest templates from the packages website and features were being added progressively on top of it. Several other models with different architectures were tested using the Optuna hyperparameter tuner. Optuna is an automatic hyperparameter optimization software library, designed for machine learning. It features define-by-run API, which dynamically construct search spaces for the hyperparameters for NN models, in the case of this work. A script written in Python featuring Optuna's tunning is provided in Annex III. It was used and generated 4 "good" models, 3 of which are informed in Table 2. The performance of the models within Optuna's framework is done by iteratively testing a range of values for the parameters of interest. The best ones are evaluated with $metric = 0.8 * loss + 0.2 * trainable$, a metric of a weighted sum of the loss function returned by the model plus a contribution of the trainable parameters of the model (the number of trainable parameters is a measure of how dense the NN is). That way, a model consisted by a small loss and a smaller model is found.

Once these parameters are coded, the training begins. During the training process, the network goes through multiple cycles known as epochs. In each epoch, the network weights each value from the data set through batches and makes predictions using forward and back propagation. The final goal of training is to minimize the error function with each epoch. By adjusting the weights and biases within the network based on the errors

[§]Here is one Tensorflow example of a similar regression problem https://tinyurl.com/59rtnyjz

observed, the network gradually improves it's accuracy in making predictions. In multiple training sessions while comparing the results, the aim is to obtain increasingly smaller errors in the between predicted and expected values.

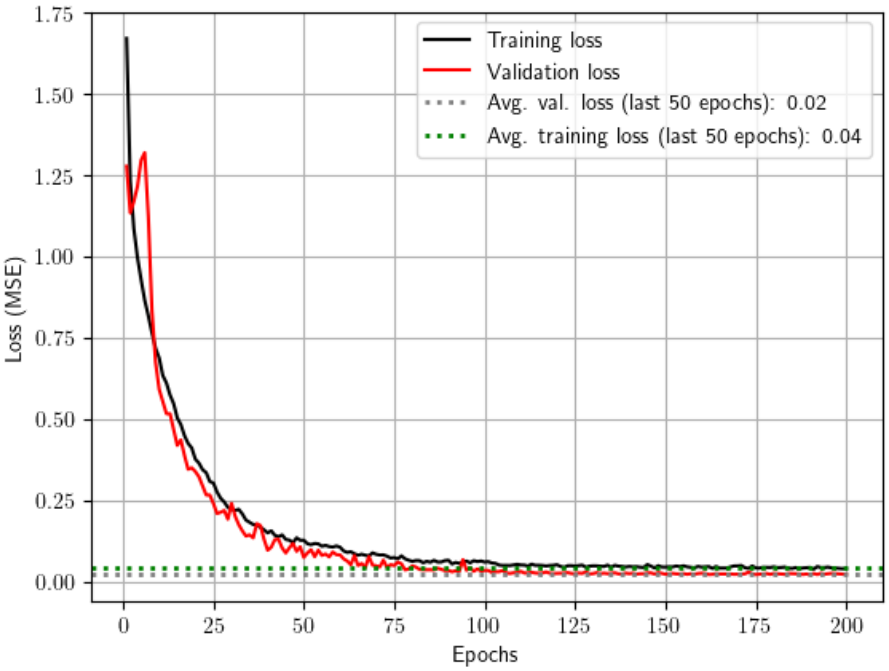
5 RESULTS

After going through the steps of collecting the data and setting up the network models one chooses to work with, there are several options to evaluate these models. Once they have been trained, the statistical evaluation of their performance depends on the type of data. The options used for this work so far have been learning curves, comparing scatter plots and heat maps. By doing so, one can address both qualitative and quantitative aspects of the work. The visualization of normal incidence photons are pretty straightforward since it contains two variables associated with a percentage. The data with angular incidence photons on the other hand leaves no alternative for visualization alike, since for those one deals with a function dependent on 4 variables that cannot be decoupled.

5.1 Photons with normal incidence

As stated in the previous section, the approach was to first train the model using photons with normal incidence. The neural network was initially trained in a personal CPU with 12GB of RAM using jupyter notebook. For the normal incidence photons with 25K points, the training process finished in about 242 seconds. The following plot in 15 shows its learning curve. Around the 100 epoch mark, the losses start to stabilize, indicating that the model was able to learn patterns in the data. The metrics used for measuring the model performance were Mean Squared Error (MSE) and Mean Absolute Error (MAE), both common functions for regression problems.

Figure 15 – Learning curve for the data set consisting of photons with normal incidence using the Keras library showing the learning process over epochs.



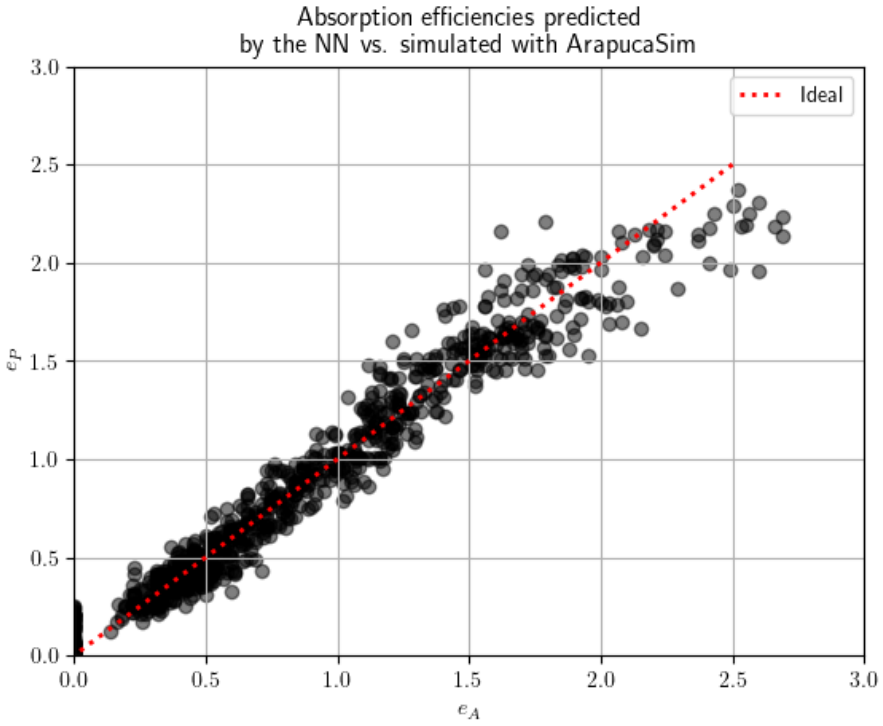
Source: The author

For this training set, the MSE was 0.072, while the MAE was 0.012. Once the NN is trained, a new unseen simulated data set can be used to test it. A plot as in Figure 16 shows the result of their correspondence. The MAE and MSE for this unseen data set were 0.011 and 0.071 respectively.

In the process of interpreting learning curves, some important aspects are considered. The validation loss curve shows how the model's performance on the validation data changes over epochs. A lower validation loss indicates better performance. If the validation loss decreases and stabilizes at a low value, it suggests that the model has learned well the data patterns. On the other hand, if the validation loss stays at a higher value or starts getting noisy separated from the overall training loss, it may indicate issues like underfitting or overfitting. Observing Figure 15, no overfitting was spot on, since both training and validation metrics decreased together. The rate at which they varied remained constant and a plateau was reached roughly at the 150-epoch mark. Furthermore, Figure 16 shows the ratio between the simulated values for new data that the NN had not seen and its predicted values. The red line represents the optimal convergence one would obtain in case the model was perfect.

The loss curve represents the training loss, in this case measured by mean squared

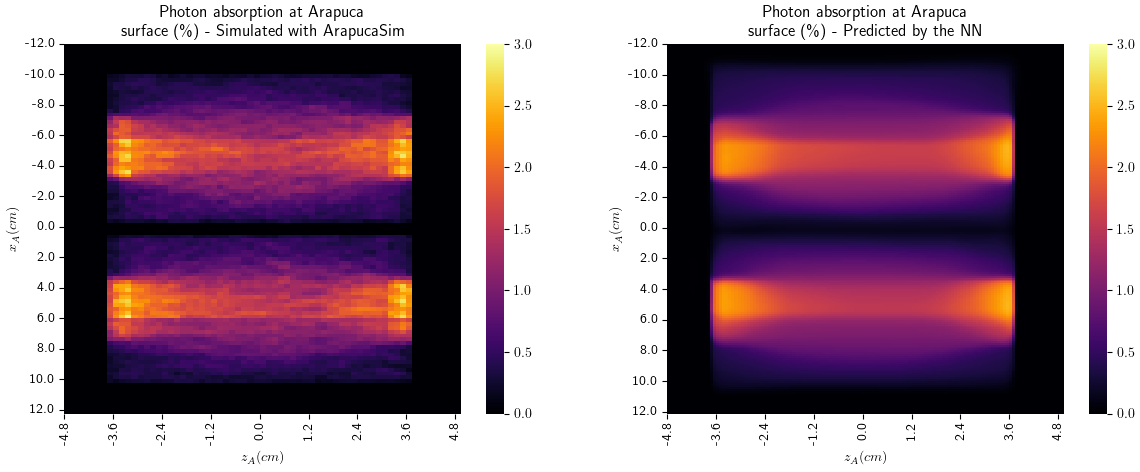
Figure 16 – Scatter plot comparing predicted absorption efficiencies (e_P) versus simulated absorption efficiencies (e_A) for normal incidence photons.



Source: The author

error, as the model iterates over the data. The loss function can be either customized by the user or get from known statistical metrics. Lower values of loss in this case indicate better fit to the training data. One might think that ideally the loss needs to decrease and converge to the lowest possible value. It would indicate that the model is learning effectively. However, it is important to pay attention to the overfitting that can occur when the model learns well on the training data but poorly on the validation. This is indicated by a discrepancy between the training and validation scores. If the loss decreases abruptly for the training data but increases for the validation part of it, it suggests overfitting. Conversely, underfitting occurs when the model fails to learn the implicit patterns in the data, leading to poor performance on both training and validation. Both were not yet observed in the results presented. Besides, if the validation score and loss curves stabilize and reach a plateau, it suggests that the model has converged and further training may not improve performance, that's why the stressed value of 200 epochs was chosen so that becomes evident. Early stopping can and was indeed utilized in this case to help prevent overfitting. Early stopping can be experimented with to see which epoch range to use in the architecture configuration.

Figure 17 – This heatmap compares the outputs of the simulation and the trained neural net efficiencies.



(a) Simulated results of ArapucaSim software for absorption efficiency by the Arapucas for photons with normal incidence.

(b) Predicted results of the trained Keras model for absorption efficiency by the Arapucas for photons with normal incidence.

Source: The author

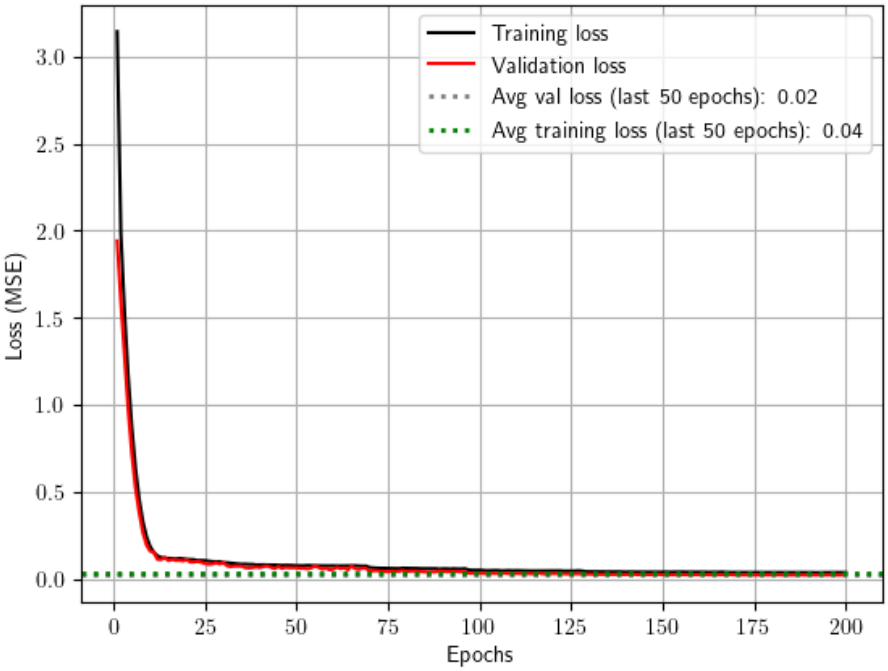
Figure 17 shows a comparison between values simulated by the ArapucaSim and values predicted with the trained NN for photons with normal incidence. Results in Figure17(a) are pixelated due to the limited number of points simulated which comes at a computational cost. On the other hand, Figure17(b) has a resolution of 4x the number of points that can be accomplished quickly using the power of the trained NN taking only 5.2 seconds.

Including the trained network within the simulation's streamline would be beneficial since the prediction for a specific incidence coordinate is considerably quicker than using the stochastic simulation employed with ArapucaSim. To avoid using the stochastic simulation within DUNE's simulation chain, tables of probabilities would be hard coded. However, for a hit at a specific point on Arapuca's surface, a look-up at the table would be required. If a specific point was not present in the table, some interpolation scheme would be applied. The trained neural network would alleviate this whole process. By imputing the hit coordinates, the network can retrieve the expected absorption with high confidence, in a short amount of time.

5.2 Photons with angular incidence

When dealing with photons with normal incidence, the complexity of the data reduces significantly, allowing for a faster and easier grasp of patterns by the neural network during the learning process. As it will be seen in the next results, once two more angle variables are included in the data set, complex and dissonant data structures show up. Figure 18 presents the learning curves obtained by the same model[¶] when applied to the data set with varying angles. The total training time taken for the angle dependency training was 4513 seconds. Anyhow, even for a more complex model, the regression training for unseen values is really fast.

Figure 18 – Learning loss curves for a data set consisting of photons with varying angles of incidence using Tensorflow.



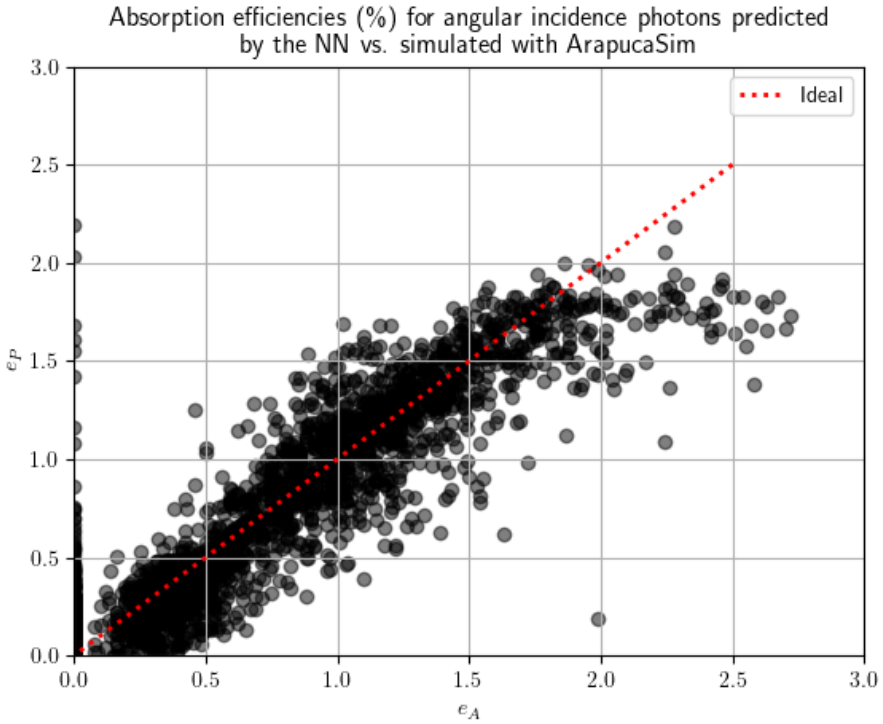
Source: The author

In Figure 19 the scatter plot exhibits the result of comparing the generated values for test data with the actual values of the simulation. Some points are visually far from where they should be. Nonetheless, convergence is still visible when analyzing the behavior of the loss curves in Figure 18. It is worth emphasizing that these points are not only errors of the NN, they are reproducing the very errors that the simulation itself produces.

[¶]The batch size of 128 instead of 64 was the only change due to the larger and time-consuming data set.

The reason for assuming that is because, as was previously mentioned, the ArapucaSim simulator generates results using the Monte Carlo approach. Although based on known models and principles, these methods generate synthetic data by repeated random sampling, therefore the data represents simulated events based on probabilistic distributions, allowing for outliers to arise.

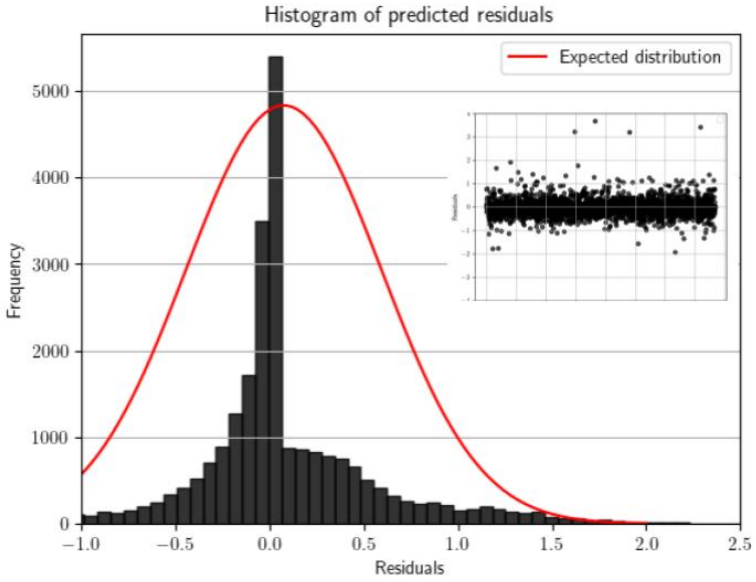
Figure 19 – Scatter plot comparing predicted absorption efficiencies (e_P) versus simulated absorption efficiencies (e_A) for angle incidence photons.



Source: The author

It is also interesting to perform residual analysis for regression models. The residuals represent the errors or the differences between the model's predictions and the true simulated values in the test data set. A histogram (Figure 20) of the residuals was created so one can visualize their frequency distribution, showing how often certain ranges of residuals occur.

Figure 20 – Visualization of how well the residuals of the model's predictions align with a normal distribution - a common assumption in statistical models.



If the residuals follow a roughly normal distribution, which is the case, the model is capturing the patterns in the data well. Source: The author

Ideally, the residuals x should fit a normal distribution curve centered around the mean value (μ) and a standard deviation (σ) of them, with gradually decreasing frequencies as it moves away from the mean, with errors being be as close as possible to 0.

6 CONCLUSIONS

The results presented has demonstrated the potential of using artificial neural networks as an efficient alternative to the Geant4-based ArapucaSim simulations for modeling the X-Arapuca device, focusing on both normal photon incidence and angles dependency. Custom datasets of absorption efficiencies were generated using the ArapucaSim to train neural network regressor architectures. By optimizing factors like the number of layers and neurons, activation functions, regularization, and learning rates, the networks were able to accurately learn the complex relationships between photon origin, incidence angles, collision locations, and absorption likelihoods for each coordinate. The neural network obtained can be exploited to analyze the X-Arapuca absorption response to a given number of photons in a given region of its surface and efficient data generation for validation purposes within the Dune Experiment.

A potential option for further enhancing the final neural network model accuracy is ensuring the training data better represents the range of possible absorption probabilities. The current discrepancy between simulated and predicted efficiencies occurs predominantly in regions of maximum absorption likelihood (>2%), where the distribution of points deviates from the straight ideal line. This may suggests an imbalance in the training data or an insufficient sampling of the high-efficiency photon incidence locations, which are less frequent. Naturally, this is a feature to be reviewed in the ArapucaSim code itself, which would constitute a parallel future work. Collecting additional simulation data points specifically for under-sampled surface areas could provide a more balanced dataset. Re-training the network on this improved data coverage may or may not improve regression performance in the peak efficiency range while not requiring new model complexities, i.e using the same models presented so far. The model could also potentially match the simulator with high precision using fewer parameters or layers by explicitly sampling rare photon interaction areas. On the other hand, the existence of extreme points not captured by the model, should not be interpreted as a defect arising from Monte Carlo methods from which the ArapucaSim is based upon. Monte Carlo methods simply generates results following a controlled probability distribution. The choice of the distribution, as well as the medium and variance values allows a greater or lesser variability in the data, whereas extreme points should emerge. In and of itself, the residual analysis is a broad

topic in statistics that encompasses inspection procedures in data sets but also formal hypothesis test. In addition to inspecting the residuals in histograms, one could test if their medium value is zero, described by a normal distribution and estimate if they are independent. Generally, in models that perfectly capture the deterministic aspect of the data generating mechanism, it is expected that the residuals come up indeed as independent, following a normal distribution, besides having zero medium value and constant variance. These observations suggest that the model calls for further development and improvement besides the presented results so far, indicating that the right approach is being followed whatsoever.

REFERENCES

1 WILLIS, W., RADEKA, V.; Liquid-argon ionization chambers as total-absorption detectors. **Nuclear Instruments and Methods**, 120(2), 221–236. 1974.

2 GRIFFITHS, D.; **Introduction to Elementary Particles; 2nd rev. version**. New York, NY: Wiley. 2008. Available at: <https://cds.cern.ch/record/111880>. Access date: 18 Jan. 2023.

3 LUBEJ, M.; Available at: <https://github.com/mlubej/standard-model/tree/master>. 2021. Access date: 20 Apr. 2023.

4 MACHADO, A.; SEGRETO, E.; ARAPUCA - a new device for liquid argon scintillation light detection. **Journal of Instrumentation**, 11(2), doi: 10.1088/1748-0221/11/02/C02004. 2016.

5 ANDREOSSI, V. et al.; X-Arapuca long term test. 2301.00420. 2023.

6 SOUZA, H.; **ARAPUCA, light trapping device for the DUNE experiment**. 2112.02967. 2021.

7 VALDIVIESSO, G.; Simulating the X-ARAPUCA, DUNE's next generation light sensors. **PoS**. doi: 10.22323/1.402.0249. Available at: <https://www.osti.gov/biblio/1870653>. Access date: 11 Aug. 2022.

8 BRIZZOLARI, C. et al.; Enhancement of the X-Arapuca photon detection device for the DUNE experiment. **IOP Publishing**. 16(9). 1748-0221. doi: 10.1088/1748-0221/16/09/p09027. 2021.

9 SMIRNOV, A.; Solar neutrinos: Oscillations or No-oscillation? 1609.02386. 2017.

10 WOLFENSTEIN, L.; Neutrino oscillations in matter. **Phys. Rev. D**. 17(9),

2369–2374. doi: 10.1103/PhysRevD.17.2369. 1978.

11 MIKHEYEV, S., SMIRNOV, A.; Resonance Amplification of Oscillations in Matter and Spectroscopy of Solar Neutrinos. **Sov. J. Nucl. Phys.**. 42, 913–917. 1985.

12 GIARNETTI, A., DAVIDE, M.; New Sources of Leptonic CP Violation at the DUNE Neutrino Experiment. **Universe**. 7(7). doi: 10.3390/universe7070240. 2021.

13 FUKUDA, Y. et al.; (Super-Kamiokande Collaboration); Evidence for Oscillation of Atmospheric Neutrinos. **Phys. Rev. Lett.**81(8), 1562–1567. doi: 10.1103/PhysRevLett.81.1562. 1998.

14 PONTECORVO, B.; Neutrino Experiments and the Problem of Conservation of Leptonic Charge. **Soviet Journal of Experimental and Theoretical Physics**. 26. 1968. Available at: <https://ui.adsabs.harvard.edu/abs/1968JETP...26..984P>. Access date: 23 Sep. 2023.

15 BUCHMUELLER, W., PECCEI, W., YANAGIDA, R.; Leptogenesis as the Origin of Matter. **Annual Review of Nuclear and Particle Science**. 55(1), 311–355. doi: 10.1146/annurev.nucl.55.090704.151558. 2005.

16 ABADI, M. et al.; TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available at: <https://www.tensorflow.org/>. Access date: 15 Mar. 2022.

17 PEDREGOSA, et al.; Scikit-learn: Machine learning in Python. **Journal of machine learning research**. 12. 2825–2830. 2011.

18 DUNE Collaboration. The DUNE Far Detector Interim Design Report Volume 1: Physics, Technology and Strategies. 1807.10334. 2018.

19 MCCULOCH, W., PITTS, W.; A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**. 5, 115–133. 1943.

20 GOODFELLOW, I., BENGIO, Y., COURVILLE, A.; Deep Learning. **MIT Press**. 2016. Available at: <http://www.deeplearningbook.org>. Access date: 27 Aug. 2022.

21 HAYKIN, S.; Neural Networks and Learning Machines. **Pearson Education**. 2009.

22 KRIZHEVSKY, A. et al.; Neural Information Processing Systems - ImageNet Classification with Deep Convolutional Neural Networks. 25. doi: 10.1145/3065386. 2012.

23 VASWANI, A. et al.; Attention Is All You Need. 1706.03762. 2017.

24 GEOFREY, H. et al.; Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. **IEEE Signal Processing Magazine**. 29(6), 82–97, doi: 10.1109/MSP.2012.2205597. 2012.

25 HOCHREITER, S., SCHMIDHUBER, J.; Long Short-Term Memory. **Neural Computation**. 9(8). 1735–1780. 0899-7667. 1997.

26 ROSENBLATT, F.; The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**. 65(6). 386–408. 1958.

27 RUMELHART, D., HINTON, G., WILLIAMS, R.; Learning representations by back-propagating errors. **Nature**. 533–536. doi: 10.1038/323533a0. 1986. Available at: <https://ui.adsabs.harvard.edu/abs/1986Natur.323..533R>. Access date: 04 Dec. 2022.

28 STANFORD UNIVERSITY 2011. Available at: <http://mlclass.stanford.edu/#:~:text=Machine%20learning%20is%20the%20science,act%20without%20being%20explicitly%20programmed.>. Access date: 12/2022.

29 CIEMAT GROUP, 2023. Available at: <https://www.ciemat.es/>. Access date: 23 Sep. 2023.

30 HEBB, D.; **The Organization of Behavior: A Neuropsychological Theory**. 9781135631901. Available at = <https://books.google.com.br/books?id=ddB4AgAAQBAJ>.

31 KANDEL, E.; **Principles of Neural Science**, Fifth Edition. 2013. Available at: <https://books.google.com.br/books?id=s64z-LdAIsEC>. Access date: 07 Nov. 2022.

32 PURVES, D. et al.; **Neurociências** - 4.ed. 9788536323756. 2010. Available at:

<https://books.google.com.br/books?id=pCtADQAAQBAJ>. Access date: 14 Dec. 2022.

33 SHEPHERD, G.; The Synaptic Organization of the Brain. **Oxford University Press**. doi: 10.1093/acprof:oso/9780195159561.001.1. 2004.

34 LECUN, Y., BENGIO, Y., HINTON, G.; Deep Learning. **Nature**. vol. 521, no. 7553. 2015.

35 DUCHI, J., HAZAN, E., SINGER, Y.; Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. **J. Mach. Learn. Res.**. 2121–2159. 2011.

36 PASZKE, A.; PyTorch: An Imperative Style, High-Performance Deep Learning Library. eprint: 1912.01703. 2019.

37 WILLIS, W., RADEKA, V.; Liquid-argon ionization chambers as total-absorption detectors. **Nuclear Instruments and Methods**. 1974. 120(2). 221–236. doi: 10.1016/0029-554X(74)90039-1. Available at: <https://ui.adsabs.harvard.edu/abs/1974NucIM.120..221W>. Access date: 10 Jul. 2023.

38 CHEN, H., LATHROP, J.; Observation of ionization electrons drifting large distances in liquid argon. **Nuclear Instruments and Methods**. 150(3). 585–588. doi: 10.1016/0029-554X(78)90132-5. 1978. Available at: <https://ui.adsabs.harvard.edu/abs/1978NucIM.150..585>. Access date: 10 Jul. 2023.

39 RUBBIA, C.; The Liquid-Argon Time Projection Chamber: A new concept for neutrino detectors. **CERN EP Internal Reports**. 77(8). 1977. Available at: <https://cds.cern.ch/record/117852/?ln=pt>. Access date: 28 Mar. 2022.

40 POWELL, C.; **The Study of Elementary Particles by the Photographic Method; an Account of the Principal Techniques and Discoveries, Illustrated by an Atlas of Photomicrographs**. 1959. Available at: <https://books.google.com.br/books?id=4l6itAEACAAJ>. Access date: 01 Jan. 2023.

41 MCGREGOR, G.; First Results from the Sudbury Neutrino Observatory. 2002.

42 HILBERT, D.; Von einer Radioansprache. 1930.

43 PESSOA, F.; **The Collected Poems of Álvaro de Campos vol. 2**. pg. 146. 2009.

ANNEXES

Annex A - A Python script for the ArapucaSim run

```
import os
import numpy as np
import csv
import pandas as pd

def angle(v1,v2): # a function to calculate the angle between two vectors
    dot = np.dot(v1,v2)
    mod1 = np.linalg.norm(v1)
    mod2 = np.linalg.norm(v2)
    cos_ang = dot/mod1/ mod2
    ang = np.arccos(cosang)
    ang_deg = np.degrees(ang)
    return ang

def theta_phi(xa,za,xg,yg,zg): # converting rectangular coordinates to angle values according to the scheme in Figure 13.
    RG = np.array([xg,yg,zg])
    RGb = np.array([xg,0,zg])
    RA = np.array([xa,0,za])
    MG = RG - RA
    pink = RGb - RA
    blue = RG - RGb
    Ref = np.array([1,0,0])
    theta_rad = angle(MG,blue)
    theta_deg = np.degrees(theta_rad)
    theta = theta_deg/360
    print("Theta (-sT input):", f'{theta:.3f}')
    print("Theta (degrees):", theta_deg)
    phi_rad = angle(Ref,pink) + np.pi
    if zg <= 0:
        phi_rad = 2*np.pi - phi_rad
    phi_deg = np.degrees(phi_rad)
    phi = phi_deg/180
    print("Phi (-sP input):", f'{phi:.3f}')
    print("Phi (degrees):", phi_deg)
```

```
35
36      return theta, phi
37
38  coordinates = pd.read_csv('coordinates.csv') # This is a separate file
        created containing the X-Arapuca coordinates (xa,za) for running the
        script
39  xa_values = np.arange(-12, 13, 1) # dimensions of the X-Arapuca cell
40  za_values = np.arange(-4.8, 5.4, 0.6)
41  results = []
42  for index, row in coordinates.iterrows():
43      xg = row['xg']
44      yg = row['yg']
45      zg = row['zg']
46      for xa_index in range(len(xa_values)):
47          for za_index in range(len(za_values)):
48              xa = xa_values[xa_index]
49              za = za_values[za_index]
50              theta, phi = theta_phi(xa,za,xg,yg,zg)
51              with open('../macros/photons.batch','r') as batch:
52                  lines = batch.readlines()
53              lines[7] = f"/gun/position {xg} {yg} {zg}\n"
54              with open('../macros/photons.batch','w') as modified_batch:
55                  modified_batch.writelines(lines)
56              command = f'./ArapucaSim -c ../data/DualCell.cfg -b ../
        macros/photons.batch -n 10000 -o result.dat -sT {theta} -sP {phi}'
57   #running the ArapucaSim using the configuration files
58              os.system(command)
59              with open('result.dat','r') as result:
60                  b = sum(line.count("B") for line in result)
61                  e = b/100
62  results.append([f'{xg:.3f}',f'{zg:.3f}',f'{xa:.3f}',f'{za:.3f}',f'{theta
        :.4f}',f'{phi:.4f}',e])
63  df = pd.DataFrame(results, columns=['xg','zg','xa','za','theta','phi','e
        '])
64  df.to_csv('output-arapucasim.csv', index=False)
```

Annex B - The Keras neural network model

```
import time
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import layers, models, callbacks

plt.rcParams['text.usetex'] = True
data = pd.read_csv('output-arapucasim.csv')
X = data[['xa', 'za', 'theta', 'phi']].values
y = data['e'].values
scaler = MinMaxScaler() # defining the normalization method
X_normalized = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
    test_size=0.2, random_state=42) # splitting the data into train and
    test

model = models.Sequential() # constructing the model
model.add(layers.Dense(4, activation='relu', input_shape=(4,),
    kernel_regularizer=tf.keras.regularizers.l2(0.001)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(512, activation='relu', kernel_regularizer=tf.
    keras.regularizers.l2(0.001)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(512, activation='relu', kernel_regularizer=tf.
    keras.regularizers.l2(0.001)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(512, activation='relu', kernel_regularizer=tf.
    keras.regularizers.l2(0.001)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1, activation='elu'))

```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
lr_scheduler = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10, min_lr=0.0001)
early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=100, restore_best_weights=True)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae', 'mse'])
ti = time.time()
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=250, batch_size=64, callbacks=[early_stopping, lr_scheduler])
tf = time.time()
Time = tf - ti
print('time taken (fit): {:.4f} seconds'.format(Time))
loss, mae, mse = model.evaluate(X_test, y_test)

new_data = pd.read_csv('output-test.csv') # loading new test data
X_new = new_data[['xa', 'za']].values
X_new_normalized = scaler.transform(X_new)
predictions = model.predict(X_new_normalized) # predictions on new data
print("Predictions for new data:")
print(predictions.flatten())
```

Annex C - An Optuna script for hyperparameter tunning

```
1
2     #example of an Optuna script for tuning the learning rate and
      dropout parameters. The same script can be used for optimizing the
      number of neurons and hidden layers, keeping the former variables
      constant or, ideally, though more computational power required, all
      four and others at the same time.
3
4  import optuna
5  import pandas as pd
6  import matplotlib.pyplot as plt
7  import numpy as np
8  import tensorflow as tf
9  from sklearn.model_selection import train_test_split
10 from sklearn.preprocessing import MinMaxScaler
11 from tensorflow.keras import layers, models, callbacks
12
13 plt.rcParams['text.usetex'] = True
14 data = pd.read_csv('test.csv')
15 X = data[['xa', 'za', 'theta', 'phi']].values
16 y = data['e'].values
17 scaler = MinMaxScaler()
18 X_normalized = scaler.fit_transform(X)
19 X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
      test_size=0.2, random_state=42)
20 lr_scheduler = callbacks.ReduceLROnPlateau(monitor='val_loss', factor
      =0.5, patience=10, min_lr=0.0001)
21 early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience
      =100, restore_best_weights=True)
22
23 def objective(trial):
24     dropout_rate = trial.suggest_uniform('dropout_rate', 0.1, 0.5)
25     learning_rate = trial.suggest_loguniform('learning_rate', 1e-5, 1e
      -2)
26     nlayers = 3
27     neurons = [512, 512, 512]
28     model = models.Sequential()
29     for i in range(nlayers):
30         if i == 0:
```

```
            model.add(layers.Dense(neurons[i], activation='relu',
    input_shape=(4,)))
        else:
            model.add(layers.Dense(neurons[i], activation='relu'))
        model.add(layers.BatchNormalization())
        model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(1, activation='elu'))
    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error',
    metrics=['mae', 'mse'])
    history = model.fit(X_train, y_train, validation_data=(X_test,
    y_test), epochs=200, batch_size=64, verbose=0, callbacks=[
    early_stopping, lr_scheduler])
    val_loss = history.history['val_loss']
    avg_loss = np.mean(val_loss[-50:])
    trainable_params = sum([tf.keras.backend.count_params(w) for w in
    model.trainable_weights])
    metric = 0.8*avg_loss + 0.2*trainable_params
    return metric

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)
best_params = study.best_params
print(f"Best parameters: {best_params}")
```