COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Coarse Energy Group Structure Optimization with Collision Probability and Artificial Neural Network Models for Reactor Analysis

Autor: Gonzalo García Gil-Delgado

Director: Luis Manuel Mochón Castro

Co-Director: Dr. Andrew Osborne

Madrid

Mayo de 2020

COLORADO SCHOOL OF MINES

Mechanical Engineering

Introduction

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título ……Coarse Energy Group Structure Optimization with Collision Probability and Artificial Neural Network Models for Reactor Analysis……

…… en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico …cuarto……. es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.:  Fecha: …26…/ …05…/ …2020…

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.:  Fecha: …15…/ …June…/ …2020…

Table of Contents

Coarse Energy Group Structure Optimization with Collision Probability and Artificial Neural Network Models for Reactor Analysis

Author: Gonzalo García Gil-Delgado

Director: Luis Manuel Mochón Castro

Co-Director: Dr. Andrew Osborne

Project Summary:

The design and analysis of nuclear reactors is frequently done using neutron transport methods that discretize continuous-energy nuclear reaction data into a finite number of energy groups. While these multigroup models can be far less computationally demanding than continuous-energy approaches, the choice of boundaries in the energy group structure can significantly affect their accuracy. The energy group structures become more important as the number of groups is reduced, and many optimization strategies have been developed to obtain an optimal group structure in a manner that does not require expert judgment. Due to the increase in computational power in recent years, machine learning algorithms are increasingly able to perform tasks that have been done traditionally by trained human operators. In this report, a multigroup collision probability code is used to generate training data for a feed-forward neural network classifier and regressor. We pre-generate cross section libraries of 20-group nuclear data with 18,034 randomly sampled group boundaries, then compute the infinite medium neutron

multiplication factors for a fixed light-water reactor geometry using the collision probability code. A feed-forward artificial neural network was used to predict the neutron multiplication factor of a given energy group structure. We found that the feed-forward neural network can predict multiplication factors with an accuracy up to 94.90% with a 5% difference between the calculated value and the prediction.

COLORADO SCHOOL OF MINES
Mechanical Engineering

Optimización de Estructuras de Grupos de Energía con Código de Colisión de Probabilidad y Modelos de Red Neuronal Artificial para Análisis de Reactores Nucleares

Autor: Gonzalo García Gil-Delgado

Director: Luis Manuel Mochón Castro

Co-Director: Dr. Andrew Osborne

Resumen del Proyecto:

El diseño y análisis de un reactor nuclear se realiza usando métodos de transporte de neutrones que discretizan datos de energía nuclear continuos en un número finito de grupos de energía. Mientras que estos modelos multigrupo son menos exigentes desde el punto de vista computacional, la selección de fronteras en los grupos de energía puede determinar su eficacia. Los grupos de energía se convierten en más importantes cuando se reduce el número de grupos, y muchas técnicas de optimización se han llevado a cabo para obtener una óptima estructura de grupos. Debido al crecimiento computacional en los años recientes, algoritmos de "machine learning" pueden realizar tareas que normalmente han sido realizadas por operadores humanos. En este proyecto, un código de colisión de probabilidad de neutrones es utilizado para generar datos de entrenamiento para red neuronal unidireccional para clasificación y regresión. El factor de multiplicación de neutrones en un reactor de agua ligera es obtenido a partir de 18,034 muestras de datos, teniendo cada muestra 20 estructuras de grupos. Una red neuronal

artificial unidireccional es utilizada para predecir el factor de multiplicación de neutrones dada una estructura de grupos aleatoria. La red neuronal ha sido capaz de predecir el factor de multiplicación de neutrones con una eficacia de hasta 94.90% con un 5% de diferencia entre el valor obtenido con el código de colisión de neutrones y la predicción.

Chapter 1. **Introduction**

This chapter introduces the background and motivation of the project in section 1.1, and the main goals and scope in section 1.2.

1.1 Background and Motivation

"Some of the ways in which energy could be generated include the use of dynamos and generators, falling water to produce hydroelectric energy, wind to produce energy using windmills and sunshine to produce energy using solar panels". One of the most common way is to use a generator combined with a nuclear reactor.

"Nuclear reactors are used as a traditional method for power generation and there are currently one hundred power plants in the US which provide roughly 20% (as shown in the figure below) of the nations' electrical needs. In other countries such as France, the production of nuclear energy can go up to 70% " [1]. There are many types of nuclear reactors but, in this thesis, I will model the Pressurized Light Water Reactor. "This kind of nuclear reactor is commonly used to fuel submarines and naval vessels and it produces 65,100 net electrical megawatts".

[1] Abdulla, "The Demise of US Nuclear Power in 4 Charts."

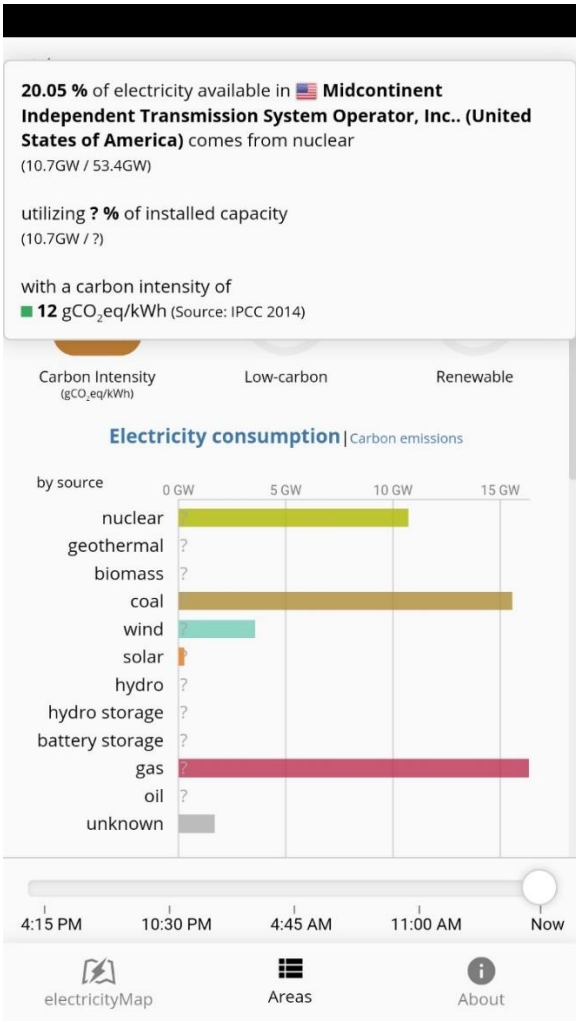

Figure 1. Percentage of Energy Generated in the Mid-Continent of the United States. *20.5% of the Mid-Continent energy comes from nuclear sources covering a total of 10.7 GW.*

Nuclear reactors are powered by a process called nuclear fission. A neutron is fired to a Uranium 235 isotope atom causing a nuclear reaction. This nuclear reaction releases large quantities of energy and causes the atom to split. New neutrons are released from the new atoms formed. Later on, we will focus on the "Neutron multiplication factor" or "K-infinity", which it is related with the number of new neutrons formed due to fission phenomenon. The new neutrons released from the split atoms impact again with new Uranium 235 atoms causing a chain reaction. The chain reaction will continue for a long period of time if not stopped. Neutrons are uncharged particles which travel following a straight line when they do not interact with other particles. Their path can be perturbed by the collision between neutrons and other matter particles. When neutrons suffer interaction with other particles, they can either be absorbed or scattered in new directions. There are many types of nuclear reactions including elastic scattering, inelastic scattering, absorption, radiative capture, nuclear fission, neutron emission and charged particles ejection. This project is based on neutron nuclear fission inside the core of a nuclear reactor. When a neutron interacts with matter, it can produce 2-3 new neutrons, light fission products and gamma rays as well as neutrinos. The heat generated by the kinetic energy of neutrons is used to heat water generating steam and making a turbine rotate to produce electricity. In order to reduce the speed of fast neutrons in the core, it is necessary to use a moderator. The neutron moderator reduces the energy of fast neutrons and leaves them as thermal neutrons, which have higher probability to cause fission inside the core.

The main structure of a Pressurized Light Water Reactor is shown below:

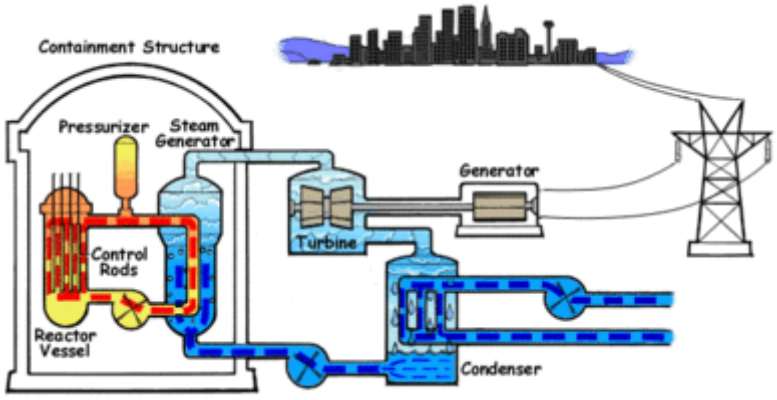

Figure 2. Representation of a Pressurized Light Water Reactor. *The two separate loops avoid the water from being nuclear polluted.*

A Pressurized Light Water Reactor has 2 separate loops. The primary loop is the one shown in the left side of the figure. In this loop the water gets heated by the nuclear reactions occurring in the reactor vessel. The pressurizer pressurizes the water up to 15.5 bar to avoid the loop from blowing when the temperature raises high. The primary loop meets a separate system known as secondary loop. This loop uses the primary loop to heat an external source of cold water and produce steam. The steam produced spins a turbine linked to a generator and we are able to produce electricity.

One major advantage of Pressurized Light Water Reactors is that it is easy to operate because less power is being produced as heat increases. But the major advantage of these types of reactors is the turbine cycle. Since both loops are separated, there is no physical contact between nuclear material and the water so it can never be contaminated. In this project, I will focus on the most important aspects to consider in a nuclear reactor: the neutron flux and the infinite neutron multiplication factor.

Modeling and simulating a nuclear reactor are key to analyze the different parameters that it has. The design and analysis of nuclear reactors is frequently done using neutron transport methods that discretize continuous-energy nuclear reaction data into a finite number of energy groups. While these multigroup models can be far less computationally demanding than continuous-energy approaches, the choice of boundaries in the energy group structure can significantly affect their accuracy. The choice of the energy group structure becomes more important as the number of groups is reduced, and optimization strategies have been developed to find optimal group structure in a manner that does not require expert judgement. The energy spectrum of the neutron field can affect the behavior of the nuclear reactor and the selected energy group structures are related with its accuracy. In this project, I predict the neutron multiplication factor based on random energy group structures.

Computational progress has allowed humans to develop approaches which combine neutron transport and material depletion to simulate the core of a nuclear reactor. In this project, we have used Monte Carlo approach to solve the Boltzmann neutron transport equation to simulate the pressurized light water reactor. Monte Carlo software provides us with high amount of accuracy and realism. However, the computational solving time can be intense. There are also multigroup simulations which are used as a way to speed up calculations. Multigroup simulations depend on the selected energy group structures and, as a result, there have been developed optimization strategies to select the energy group structures correctly. The energy group structures selected can affect the accuracy of the predictions of the nuclear reactors' parameters. These methods include the manual selection of group structures until achieving a desired

accuracy. More algorithms have been used to predict the group structures correctly such as "SIMMER extension for multigroup energy structure search using genetic algorithm with different fitness functions" [2] and "particle swarm optimization" [3].

"Computational evolution allows us to solve problems using machine learning algorithms which can learn of a problem following a certain pattern. Computers can be fed with large amounts of data and be able to learn, analyze and make conclusions. Machine learning has applications in all kinds of industries which include energy, retail, healthcare and life sciences, travel and hospitality, financial services" [4]. In this project, an artificial feed forward neural network is used as a machine learning algorithm to predict the infinite neutron multiplication factor of a certain energy group structure, leading to a reduction in computational time compared to multigroup models and Monte Carlo approach.

1.2 Goals and Scope

Due to the increase in computational ability, machine learning algorithms have been increasingly used to replace optimization strategies. In this report, a multigroup collision probability code is used to generate training data for a feed-forward neural network classifier and regressor. We pre-generate cross section libraries of 20-group

[2] Massone, Gabrielli, and Rineiski, "SIMMER Extension for Multigroup Energy Structure Search Using Genetic Algorithm with Different Fitness Functions."
[3] Yi and Sjoden, "Energy Group Structure Determination Using Particle Swarm Optimization."
[4] "What Is Machine Learning (ML) and Why Is It Important?"

nuclear data with 18,034 randomly sampled group boundaries, then compute the infinite medium neutron multiplication factors ($K_{infinity}$) for a fixed light-water reactor geometry using the collision probability code. A feed-forward artificial neural network was used to classify the group structures according to whether the calculated infinite medium neutron multiplication factor was accurate relative to a 500-group benchmark calculation. As a result, we found that, using 18,034 data samples, of which 16,231 were used for training, the feed-forward neural network can predict the classification of a group structure with an accuracy of 94.2%.

A modified artificial feed-forward neural network was used to predict the neutron multiplication factor given a certain group structure. Instead of using the multilayer perceptron classifier, we used the multilayer perceptron regressor. It is relevant to analyze and optimize the artificial feed-forward neural network in order to predict the neutron multiplication factor given a group structure. The collision probability code developed by Dr. Osborne group research contains complex algorithms to simulate a real nuclear reactor. With the collision probability code, we are able to feed the reactor with a random group structure and obtain useful parameters from the reactor, including the neutron multiplication factor. However, a high-performance computing clustering, Aun, is needed to run the collision probability code and the running time is still not very efficient. The purpose of the neural network is to learn and predict the parameters of the nuclear reactor given a random group structure as the simulation takes such a lower amount of time. In this thesis we will focus on predicting the neutron multiplication factor.

The particular tasks that I performed in this project are the following:

1. Generation of the training set. Using the collision probability software developed by Dr. Osborne group research, generate 10,000 sets of cross section libraries of a given group structure and use this as input to the collision probability model which will generate fission and capture reaction rates (as well as the infinite neutron multiplication factor
2. Develop the feed forward artificial neural network. I used Scikit-Learn to train a feed forward artificial neural network using random group structures as inputs and the infinite neutron multiplication factor as output.
3. Conduct a systematic optimization of the neural network to maximize its classification and regression performance using standard methods.

Chapter 2. Technology Description

This chapter introduces the technology used for this thesis. In particular, explaining the collision probability code to model nuclear reactors in section 2.1, the artificial feed forward neural network in section 2.2 and the methods I used to optimize the feed forward artificial neural network in section 2.3.

2.1 Collision Probability Model and Neutron Flux

Neutron flux is a scalar value which is significantly important when it comes to analyze the efficiency and correct functionality of nuclear reactors. It is the total length traveled by all free neutrons per unit time and volume. "Another way we could define the neutron flux is to imagine a sphere of radius R and compute the number of neutrons which go through the cross section of the sphere" [5]. In order to know the thermal power and efficiency of a nuclear reactor is necessary to know how many neutrons are traveling through the core. To evaluate this measurement, we define a parameter called "Neutron Density" (n) with units [$neutrons/cm^3$], which gives us the amount of neutrons in a cubic centimeter. If we want to know the number of neutrons passing through a cross

[5] "Neutron Flux."

section it is relevant to define their velocity as a parameter (v). The neutron flux density is calculated as follows:

$$\Phi = n \cdot v$$

Where the parameters and its units are shown below.

$\Phi \equiv neutron\ flux\ [\frac{neutrons}{cm^2 \cdot s}]$

$n \equiv neutron\ density\ [\frac{neutrons}{cm^3}]$

$v \equiv neutron\ velocity\ [\frac{cm}{s}]$

The neutron flux is a scalar and provides us with the information of the number of neutrons passing through a certain cross-sectional area in all directions per unit volume. In order to determine the rate interactions of neutrons with matter, it is worth considering the graph "Neutron flux vs Energy" as the nucleus reactions depend on energy. In the case of this thesis, the plot neutron flux vs the energy of incoming neutrons will be in the range: [1mEv-10Mev] because that captures most of the energy spectrum without sacrificing a lot of resolution. This plot is generated with the collision probability code developed by Dr. Osborne research group. The flux is generated by the model depending on user defined parameters. Using group structures energy as inputs to the collision

probability code will give us the flux and we are able to plot the graph "Neutron flux vs Energy".

"The collision probability code V:BUDS (Visualize: Burnup, Depletion, Spectrum) was first used to analyze fuel cycle systems" [6]. This code can simulate the neutron transport and depletion of 24 actinides and burnable absorbers in a cell consisting of two homogenous mediums. A new model has been developed in order to perform simulations in two cells (VBUDSII). "Also, a new depletion capability was implemented in VBUDSII using Massachusetts Institute of Technology's depletion module, OpenMC. The last version of the modified collision probability code stands for VBUDS3, which is the one that has been used in this project and has the ability to track a total of 255 nuclides including 42 actinides and 150 fission products, as well as activation products and burnable absorbers. VBUDS3 has been implemented in MATLAB and Python softwares and depends only on ENDF data, NJOY2016 and OpenMC". To run the collision probability code, or VBUDS3, it is necessary the use of a supercomputer as it is a very complex model and would take a lot of time to run in a normal computer. I have been given access to a Colorado School of Mines cluster in order to run the analysis. The collision probability code receives random energy group structures as inputs, as well as the nuclear cross section graphs. VBUDS3 generates a collapsed cross section chart, which is used afterwards to generate the neutron flux with respect to the energy of the incoming neutron. The collision probability code outputs the infinite neutron multiplication factor and the neutron flux. They will be used to generate the inputs of the feed forward neural network.

[6] Berry and Osborne, "A COLLISION PROBABILITY AND DEPLETION MODEL FOR RAPID SCOPING AND OPTIMIZATION OF NUCLEAR REACTORS."

2.2 Feed Forward Artificial Neural Network

I have used a Feed Forward Artificial Neural Network and trained it to predict the neutron multiplication factor ($K_{infinity}$) given a random energy group structure. This is useful because the neural network can predict the neutron multiplication factor of a given group structure and avoid using the complex collision probability model and Monte Carlo continuous energy approach. Also, it is faster and, consequently, more computationally efficient.

"Artificial neural networks can be imagined as human brains. The network is fed with training data and asked to do a specific task and then it is tested with a testing set that has not been seen before and it learns to perform the task. Increasing the number of data in the training set allows the neural network to learn more and more about certain topic" [7]. In this thesis I have been modifying the testing set of data between ten and thirty per cent to check the neural network's performance, as well as the number of neurons in each layer and the solver type. "The best neural networks are used by banks to process checks and by post offices to recognize addresses". Another example of the application of neural networks is in image recognition. A neural network can be asked to identify certain objects in an image and not knowing even what that object is.

"There are many types of neural networks such as feed forward neural network, radial basis function neural network, recurrent neural network, modular neural network". But in this thesis, I have used the Feed Forward Artificial Neural Network which takes inputs

[7] "Artificial Neural Network."

and releases outputs without making cycles between nodes. It is the simplest and oldest neural network and the information flows in one direction (Forward). The neural network consists of layers. The first layer is called the input layer and the last one the output layer. The layers in between are called hidden layers. A general feed forward neural network structure looks as follows:

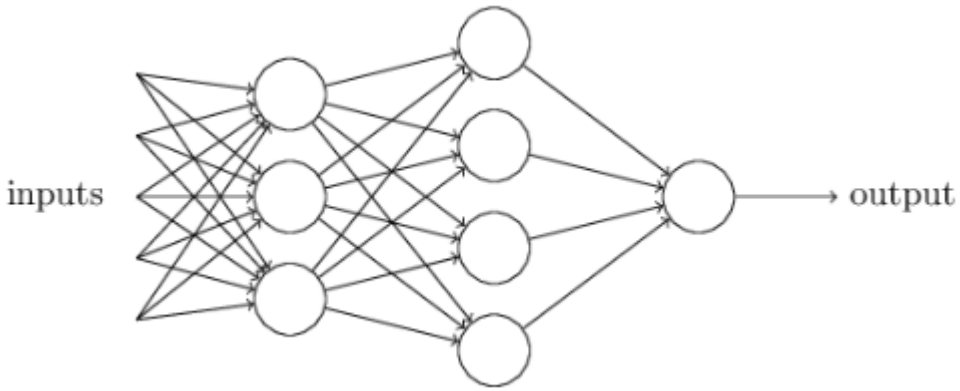

Figure 3. Generic Structure of a Feed Forward Neural Network. *The information flows in one direction and the network consists of one input layer, one output layer and user defined hidden layers.*

In this project I have used the random energy group structures as input to the feed forward neural network and the neutron multiplication factor as the output layer. Each layer is made of neurons which receive an input and provide an output as seen in the below figure.

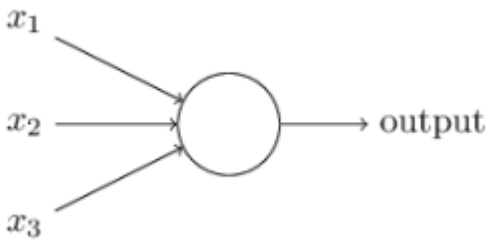

Figure 4. Basic Structure of a Neuron. *Each Neuron Has Binary Inputs Which Generate an Output Depending on Its Bias and Input Weights.*

Usually, the inputs in the neuron are binary numbers and the output follows the below equation.

$$Output = 0 \; if \sum_j w_j \cdot x_j \leq Threshold$$

$$Output = 1 \; if \sum_j w_j \cdot x_j > Threshold$$

Where w_j represents the weights, which can be understood as the importance that the neural network gives to an input." Each neuron of the network has its own bias, which can be understood as the easiness for the neuron to output a 1".

$$output = \begin{cases} 0 \; if \; w \cdot x + b \leq 0 \\ 1 \; if \; w \cdot x + b > 0 \end{cases}$$

A neuron with a really high bias will output a 1 very easily. "Small changes in the weights and biases of the neurons can result in small changes in the outputs" [8]. In the example shown above, each neuron of the input layer is making three simple decisions, which are based on the input evidence." The neurons in the second layer make a more important decision than those of the first layer by weighting up the results provided by the first layer of neurons". Each neuron has only one output, which is used as input for the following neurons in the next layer. Multilayer perceptron methods have been implemented to model the structure of feed forward neural networks but, the number of hidden layers and neurons in each layer is obtained based on hit and trial methods.

"The main goal of the feed forward neural network is to minimize the value of the cost function. This function outputs the difference between the target value we want to reach, and the approximation made by the network" [9]. The cost function of the feed forward neural network can also be written in many ways, being the mean squared error function one of the many functions used which follows the below equation.

$$C_{MST}(W, B, S^r, E^r) = \frac{1}{2} \cdot \sum_j (a_j^L - E_j^r)^2$$

[8] Nielsen, "Neural Networks and Deep Learning."
[9] Upadhyay, "Feed Forward Neural Networks."

"Where W is the weights of the neural network, B is the biases, S^r is the training sample input and E^r is the desired output of the training sample". The input layer of neurons provides the network with initial values and, with each value there is a weight associated. The activation function of a neural network determines the output of each neuron given an input. There are many types of activation functions including RELU, TanH, Sigmoid, Identity. Each activation function used in neural networks have different effects in the outputs. Our objective is to find the most suitable activation function to the feed forward artificial neural network which produces the most accurate output.

Machine learning techniques allow us to develop algorithms for many applications. Including classification and regression. In artificial neural networks, the output could be either a binary variable or a real number. If the output is a real number, the neural network would be performing a regression algorithm. Regression models are implemented as multilayer perceptron networks, which is the case in this project." Multi-layer Perceptron is a supervised learning algorithm which has some advantages included the capability to learn non-linear models and capability to learn models in real-time (on-line learning)" [10].

Python is a programming language which can be very useful to model machine learning algorithms. In this case, I have used Scikit-learn machine learning library for Python. "Scikit-learn has ability to perform various algorithms such as vector machine, random forest, k-neighbours". The regression model implemented in the project is a supervised learning problem. The training data comes with certain parameters which we want to predict, in particular, the infinite neutron multiplication factor.

[10] "1.17. Neural Network Models (Supervised) — Scikit-Learn 0.23.0 Documentation."

Chapter 3. Model's Description

This chapter introduces the description of the model. The model's specifications in section 3.1, the data used in section 3.2 and the algorithms in section 3.3

3.1 Specifications

The nuclear reactor's neutron field and material composition are key tools to design and analyze how it behaves. They determine the stability of the nuclear reactor and burnup performance, as well as internal variables worth considering such as temperature, radiation damage, chemical changes and structural stresses. The neutron energy spectrum has significant effect in these properties. There have been developed computational approaches which combine neutron transport and material depletion in order to simulate the core of a nuclear reactor. There are also other approaches which include multigroup codes that perform simulations in a discretized way and can be computationally faster. In this project, we implemented a discretized model of Monte Carlo software to solve the Boltzmann neutron transport equation. Monte Carlo simulations can predict fairly well the behavior of neutron transport depletion. Individual neutrons from source points are generated using the Monte Carlo approach, drawn from an initial guess of their distribution. The neutrons generated have random start points and follow random directions. In continuous energy models, many movements of each neutron have to be considered and, consequently, a lot of probabilistic models need to be considered. The

collision probabilities of neutrons depend on the geometry of the core of the nuclear reactor. In this project, the reactor's core has infinite Lattice geometry made of pins and antipins surrounding the corresponding pins. The pins contain fuel and the surrounding pins contain coolant, in this case water. There are many types of collision probabilities which are needed in order to simulate the movement of neutrons as accurate as possible. The scape and transmission probabilities depend on the geometry of the cell and the total cross section. The scape probability represents the probability of a neutron leaving cell i without having any interaction which had its last interaction in that cell, either absorption or scattering. The transmission probability is the probability that a neutron has to leave cell i without having interacted through scattering or absorption with matter. The main difference between the two probabilities is how neutron i entered the cell." Along with these collision probabilities, there are scattering angle probabilities, rotational approximations, self-collision, reciprocity" [11]. In this project, we will consider neutron fission as the most relevant for our purpose.

The Watt function represents the distribution of the number of neutrons with energy in the fission spectrum. The fission spectrum is different for each nuclide. U235 has the following fission neutron spectrum:

[11] Deinert and Schneider, "A Multi-Region Collision Probability Method For Determining Neutron Spectra and Reaction Rates."

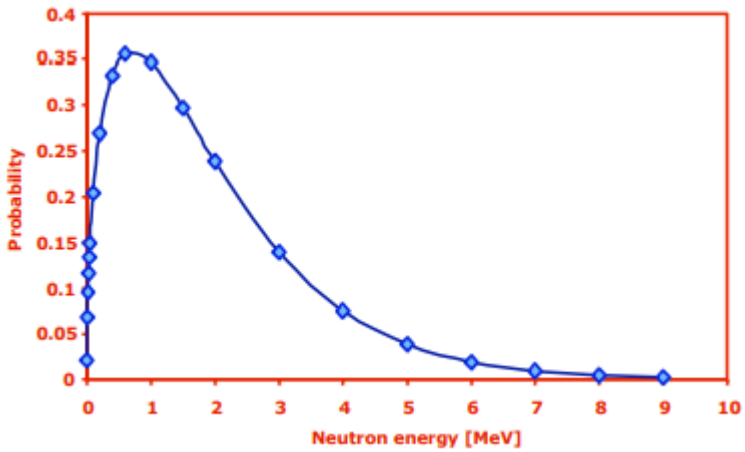

Figure 5. Fission Spectrum for U^{235}Nuclide. *Probability of fission of each incoming neutron of U^{235} with respect to the neutron's energy.*

The selected neutron interaction using continuous energy approaches is random but depends on the cross section of each interaction in a given material." The probability of a neutron from fission having an energy between E and E+dE is the function P(E)dE "[12].

$$P(E) = 0.4865 \cdot \sinh\left(\sqrt{2 \cdot E} \cdot e^{-E}\right) \ [MeV^{-1}]$$

[12] Watterson, "The Watt Distribution (Spectrum)."

Neutrons are tracked until they leave the region of interest (the core of the nuclear reactor), which properties are defined by the user. This process of tracking neutrons is repeated until there is enough data to carry out a statistical study. Monte Carlo approach can reach incredible levels of accuracy and realism. However, the main disadvantage is the computational requirements. They are usually very intense and sometimes it is needed billions of iterations to ensure accuracy. Monte Carlo methods use codes such as MCNP or Serpent to solve the neutron transport equation and is combined with packages such as ORIGEN to implement depletion simulations in the nuclear reactor.

VBUDS3 was used as a discretized way to simulate a pressurized light water reactor using Serpent 2 to simulate burnup and depletion capability. VBUDS3 uses the same collision probability method used in VBUDSII which divides the core of the nuclear reactor in N fuel pins and coolant regions. Neutron flux and cross sections of nuclides are assumed to be uniform along each region. Using infinite lattice, it can be shown that the neutron flux Φ_g^i [1/b-s] in each energy group g and region I must satisfy the following equation:

$$V^i \cdot \sum_{t,g}^{i} \Phi_g^i = \sum_{j}^{N} V^j \cdot p_g^{ji} \cdot \left[\sum_{g'} \sum_{s,g'g}^{j} \Phi_{g'}^j + \frac{1}{k} \cdot \chi_g \cdot \sum_{g'} \nu \cdot \sum_{f,g'}^{j} \Phi_{g'}^j \right]$$

Where V^i represents the volume of region i $[cm^3]$, $\sum_{t,g}^{i} \Phi_g^i$ is the total cross sections [1/cm], $\sum_{f,g'}^{j} \Phi_{g'}^j$ is the fission cross sections [1/cm], $\sum_{s,g'g}^{j} \Phi_{g'}^j$ is the scattering kernel

from group g' to group g [1/cm], while χ_g , k and ν are the fission spectrum, the neutron multiplication factor, and the neutron multiplicity per fission.

The value of p_g^{ji} is the probability that a neutron has to appear in group g and region j after a collision or reaction will have its next collision in region i. The value of this probability depends on many factors including the geometry and composition of the region and the neutron energy, as well as the transmissions and escape probabilities for each region. The equation shown above can be expressed as a system of equations:

$$L \cdot \Phi = \frac{1}{k} \cdot F \cdot \Phi$$

Where L represents a matrix of removal minus scattering reactions and F represents a matrix of fission reactions. MATLAB's eigenvalue library is able to solve the eigenvalue problem and obtain the multigroup neutron flux.

Materials inside the core of the nuclear reactor evolve following Bateman equation:

$$\frac{d\,N_i}{dt} = \sum_{j \neq i} N_j \cdot \left[\sigma_{f,j} \cdot a_{ji} \cdot \Phi + \sigma_{c,j} \cdot b_{ji} \cdot \Phi + \lambda_j \cdot c_{ji}\right] - N_i \cdot \left[\lambda_i + \sigma_{a,i} \cdot \Phi\right]$$

The value of N_i is the atomic number density of nuclide i [$1/cm^3$], $\sigma_{f,j}$ is the fission cross section of nuclide j [barns] and $\sigma_{c,j}$ is the neutron capture cross section of nuclide j [barns]. Also, the yield of nuclide I produced by fission in nuclide j is represented with the symbol a_{ji}. b_{ji} gives us the probability that reactions in nuclide j produce nuclide i. The neutrons absorbed reactions in nuclide i are represented by $\sigma_{a,i}$. Nuclide j's decay constant is λ_j, which has a branching ratio to nuclide i of c_{ji}. Bateman equation can be expressed with matrices as follows:

$$\frac{d\,N}{dt} = (A \cdot \Phi + B) \cdot N$$

While A is the neutron reaction matrix [barns] and B is the decay matrix [1/s], N is the vector of atomic number densities [$1/cm^3$]. The transmutation matrix $= (A \cdot \Phi + B)$ are automatically created with the depletion module of OpenMC with the user specifying XML representation of depletion chain. In this project, CASL depletion chain was used from OpenMC package. VBUDS3 computed internal reaction rates using the multigroup neutron flux, ENDF data and a power density defined by the user [13].

The simulations performed using VBUDS3 were applicable to a pressurized light water reactor using four nuclides (U235, U238, H1 and O16). Also, the software uses water moderator in and infinite lattice of fuel pin cells. The modeled pin cell of the nuclear reactor's core is shown in the following figure:

[13] Berry and Osborne, "A COLLISION PROBABILITY AND DEPLETION MODEL FOR RAPID SCOPING AND OPTIMIZATION OF NUCLEAR REACTORS."

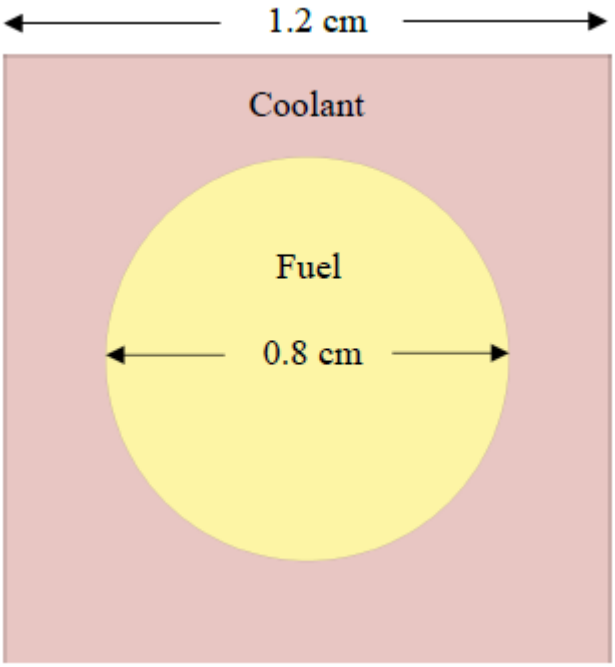

Figure 6. Unit pin cell of a light water reactor simulated in VBUDS3. *The fuel consisted of Uranium Dioxide at an enrichment of 3 a/o, with water as the coolant.*

The specifications of the simulated pressurized nuclear reactor are shown in the table below.

Parameter	Value
Fuel Volume [cm^3]	0.50
Moderator Volume [cm^3]	1.46
Fuel Density [g/cm^3]	10.8
Moderator Density [g/cm^3]	0.72
Pin Pitch [cm]	1.2
Pin Radius [cm]	0.4
Fuel Temperature [K]	294
Moderator Temperature [K]	294

Table 1. Simulation Parameters Osborne, Timothy A. Smith, and Mark R. Deinert, "Comparison of Actinide Production in Traveling Wave and Pressurized Water Reactors."

The development of multigroup models like VBUDS3 allows us to solve complex continuous problems involving nuclear reactors using discretized approaches. Multigroup models are computationally more efficient and the approximations can lead to accurate results. The objective sought to optimize VBUDS3 is obtain a reasonable amount of energy group structures which can lead to accurate results. It has been observed that a model could be simulated with just only 20 energy group structures. In order to understand the goal, the following image illustrates a project of discretizing the neutron

flux spectrum [14]. However, in the following example, there are many groups which it would be suitable to minimize and reduce computational cost getting the most accurate results as possible. The example illustrates how a spectrum can be made using discretized methods.

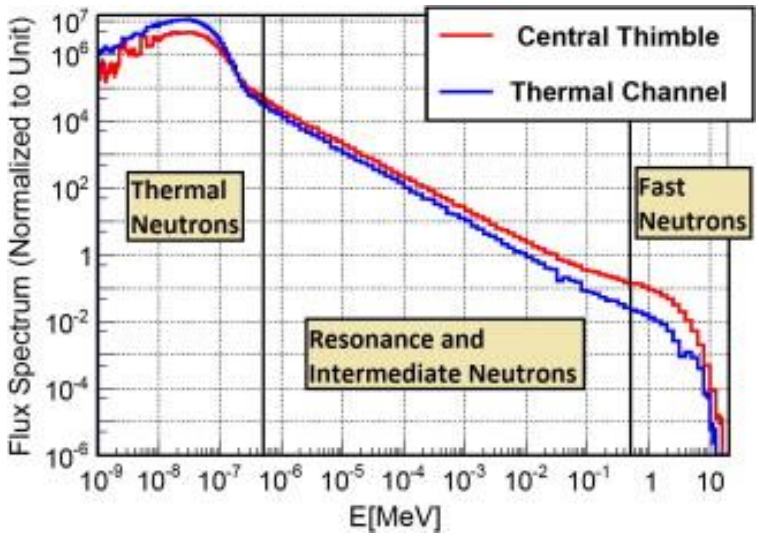

Figure 7. Generic Example Illustrating How a Neutron Flux Can Be Discretized.

[14] Chiesa, Previtali, and Sisti, "Bayesian Statistics Applied to Neutron Activation Data for Reactor Flux Spectrum Analysis."

In this project, the energy group structures are randomly selected between the energy range 0.001 eV and 10MeV. This range is limited because these limits capture most of the energy spectrum. It has been observed that with only 20- group energy structures, the nuclear reactor can be modeled. The random energy group structures are selected using the cross sections of the different nuclides used in the project. These random energy group structures along with cross section spectra are used as input to VBUDS3. Cross section spectra of the nuclides U235, U238, H1 and O16 are used to simulate the pressurized light water reactor. The collision probability code is able to compute a collapsed discretized cross section spectrum to generate the also discretized neutron flux. However, this collapsed cross section is only graphed using previously the neutron flux, which is still unknown. Several approximations must be made for VBUDS3 to obtain the neutron flux based on the collapsed cross section spectrum. Narrow resonance approximation assumes that the flux follows equation $\frac{1}{E}$ when the resonances of the collapsed cross section spectrum are narrow. However, as it can be seen in the different nuclides cross section spectra, there are some energy ranges where the resonances are not narrow enough, and the narrow resonance approximation is not applicable.

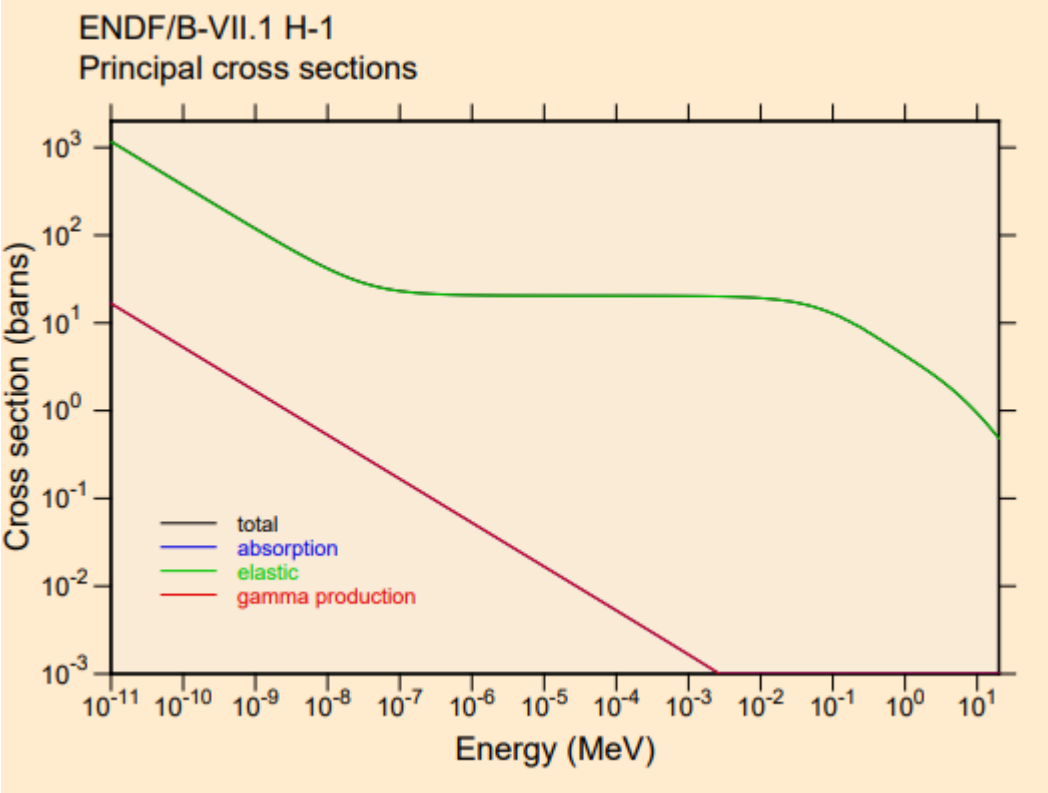

Figure 8. ENDF/B-VII.1 Incident-Neutron Data. *Cross Section of* H^1 *nuclide.*

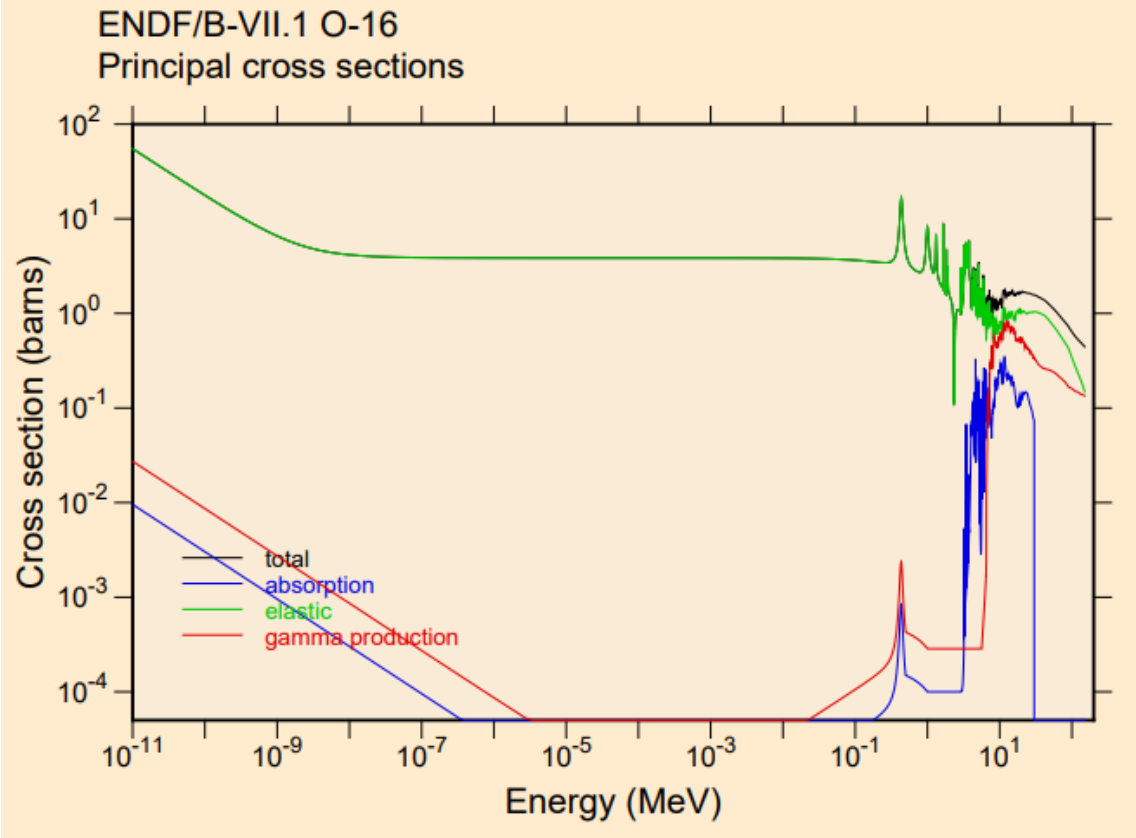

Figure 9. ENDF/B-VII.1 Incident-Neutron Data. *Cross Section of* 0^{16} *nuclide.*

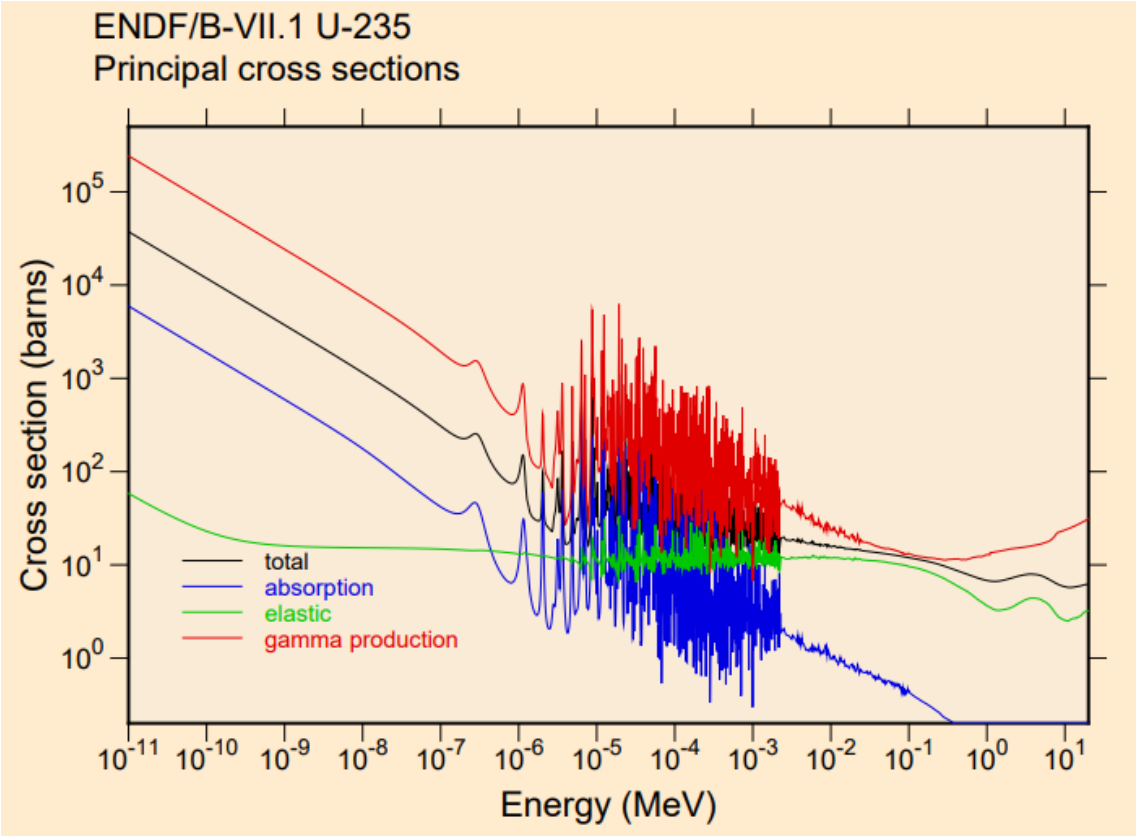

Figure 10. ENDF/B-VII.1 Incident-Neutron Data. *Cross Section of* U^{235} *nuclide.*

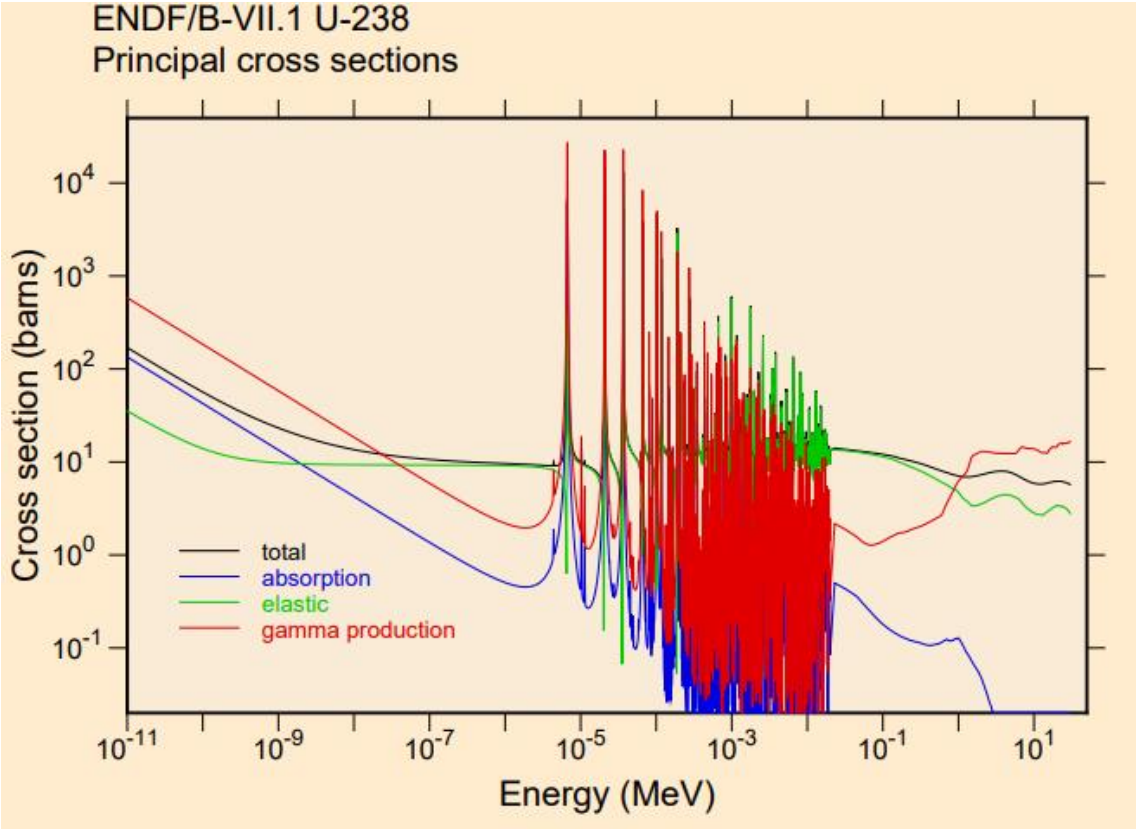

Figure 11. ENDF/B-VII.1 Incident-Neutron Data. *Cross Section of* U^{238} *nuclide.*

In order to make the same approximation, we have to ensure that there are enough energy bins where the resonances are not narrow enough. Once the collapsed cross section spectrum is created, VBUDS3 uses it to compute the discretized neutron flux and consequently, output the parameters of the nuclear reactor. In this project, I used the neutron multiplication factor as the relevant parameter of the pressurized light water reactor. VBUDS3 performs this task in a reasonable speed. In part, this is because there are only four nuclides used in the simulation our nuclear reactor. However, the simulation with VBUDS3 can be challenging when dealing with more complex systems involving more nuclides and matter. This is the reason why I set up an artificial feed forward neural network. Problems in the future can be more challenging and machine learning allows us to solve more complex problems, including nuclear reactor of different types and with more nuclides and matter involved.

A feed forward artificial neural network was used as a machine learning tool to predict the infinite neutron multiplication factor given a random energy group structure. The neural network enables us to feed it with a randomly generated energy group structure with 20 groups and it will predict the k_{∞}. Although VBUDS3 does not require a lot of computational effort in this case due to the simulation of the nuclear reactor with only four nuclides, the training of an artificial neural network is handy because it can be used in future and more complex models. The feed forward neural network has been developed using scikit-learn package of Python and it is modeled using a multilayer perceptron regressor (MLPRegresor). There are many inputs which we have to select in order to model the neural network using regression. Hyperparameters of a neural

network are those which cannot be learned, and the user has to specify them in order to obtain optimal results.

The learning rate of an artificial neural network enables us to control how much the weights and biases are adjusted with respect to the loss [15]. It tracks the slowness traveled along the downward slope. If the learning rate is too low, we are able to make sure that we do not miss any local minima but, the convergence could take time. It is a task for the user to configure the learning rate such that it is not too high or too low. A very high learning rate can lead to increase the loss and the model could fail to converge or even diverge, while a nice selection of the learning rate leads to a decrease. The learning rate hyperparameter is set to be "adaptive" in this model. Depending on the selected learning rate, the neural network will update the weights and biases in a different manner. The adaptive learning rate keeps the learning rate to the initial value (set to default equal to 0.001) as long as training loss keeps decreasing [16]. If two consecutive epochs do not decrease training loss by a stablished tolerance (10^{-5}), the learning rate is divided by five.

The hidden layer size is another hyperparameter of the neural network which cannot be learned through training the model. The number of hidden layers and the number of neurons in each layer constitute a hyperparameter of great importance in the neural

[15] Zulkifli, "Understanding Learning Rates and How It Improves Performance in Deep Learning."
[16] "Sklearn.Neural_network.MLPRegressor — Scikit-Learn 0.23.0 Documentation."

network. These values are tuned manually and depending on the number of hidden layers and neurons in each layer the results are different. Optimization strategies involving these hyperparameters have been applied in this project, such as grid search, random search and simulated annealing. For simplicity, there has been developed a grid of values containing two hidden layers and the same number of neurons in each layer. Using that grid to estimate the best accuracy of the model, there are some fixed parameters defined by the user which include the activation type of the network, the solver type and alpha. The number of hidden layers is an input to MLPRegressor and was determined using grid search and analyzing which hidden layer size produced the best results. Values of the number of hidden layers and neurons in each layer are shown in the "Results" section.

The solver type of artificial neural networks represents the optimization strategy we are selecting for our problem. The solver used to model the feed forward neural network was picked depending on the accuracy of the output. Lbfsgd, Adam and sgd are the solvers which have been analyzed for the purpose of the project (predicting the infinite neutron multiplication factor). Later in the paper it will be explained using graphs which solver type produced the most accurate results but, generally speaking, 'lbfgs' was the most accurate. 'Lbfgs' is an optimizer in the family of quasi-Newton methods. However, the running time was not as fast as 'adam' solver type.

I have tested the feed forward artificial neural network using every possible solver type. 'Relu' is a type of activation function which is commonly used in neural networks and stands for rectified linear unit [17], defined as $y = \max(0, x)$ and looks as figure below:

[17] Liu, "A Practical Guide to ReLU."

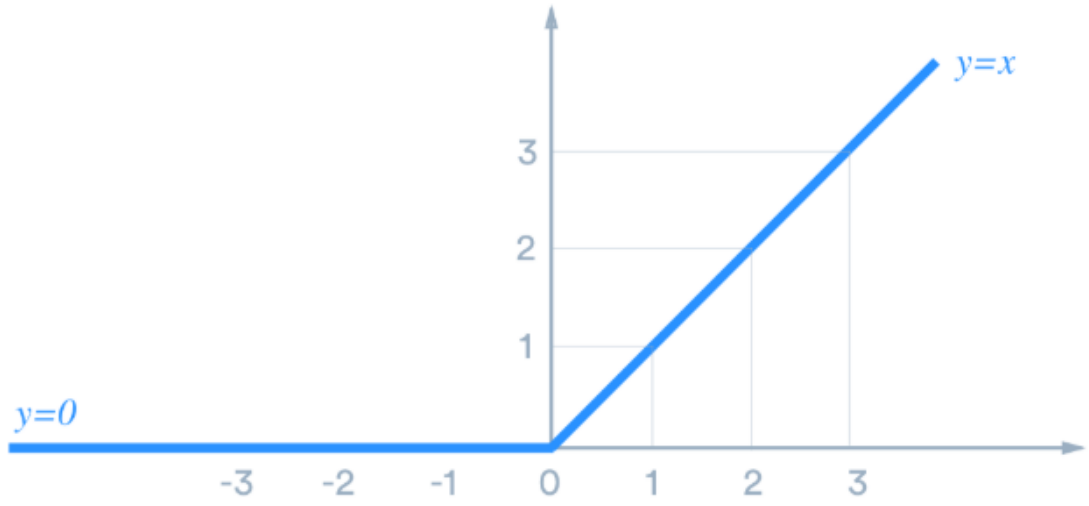

Figure 12. Relu Activation Function. *Positive linear activation function.*

Because the function is linear, there is no complicated math behind 'relu' activation function, and it is appropriate to use it in our feed forward neural network." The rectified linear activation function is used as the default activation function in many artificial neural networks because it is easy to train and can achieve better performance" [18]. However, each artificial neural network is different from one another and in this case the activation function which performed best was 'tanh'. 'Tanh' activation function is defined as follows:

[18] Brownlee, "A Gentle Introduction to the Rectified Linear Unit (ReLU)."

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

"The nature of this activation function is nonlinear, which makes the neural network to perform slower and its bounds are in the range between -1 and 1" [19]. 'Tanh' activation function provides the best accuracy in the modeled feed forward neural network, which means that each neuron is fired using tanh function, but the neurons in early layers learn very slowly compared to the ones in the output layer.

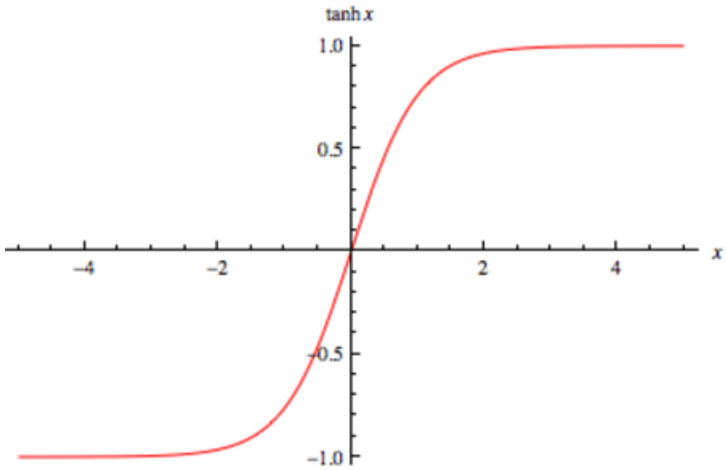

Figure 13. Tanh Activation Function. *The neurons in the feed forward neural network are activated with this activation function.*

[19] Jain, "Complete Guide of Activation Functions."

In the opposite side, testing the feed forward artificial neural network using 'identity' activation function, it was found that the performance was not very accurate. Also, a disadvantage of this type of activation function in neurons is that "all the layers of the network are collapsed in one". Although the structure becomes more simplified, the accuracy begins to decrease.

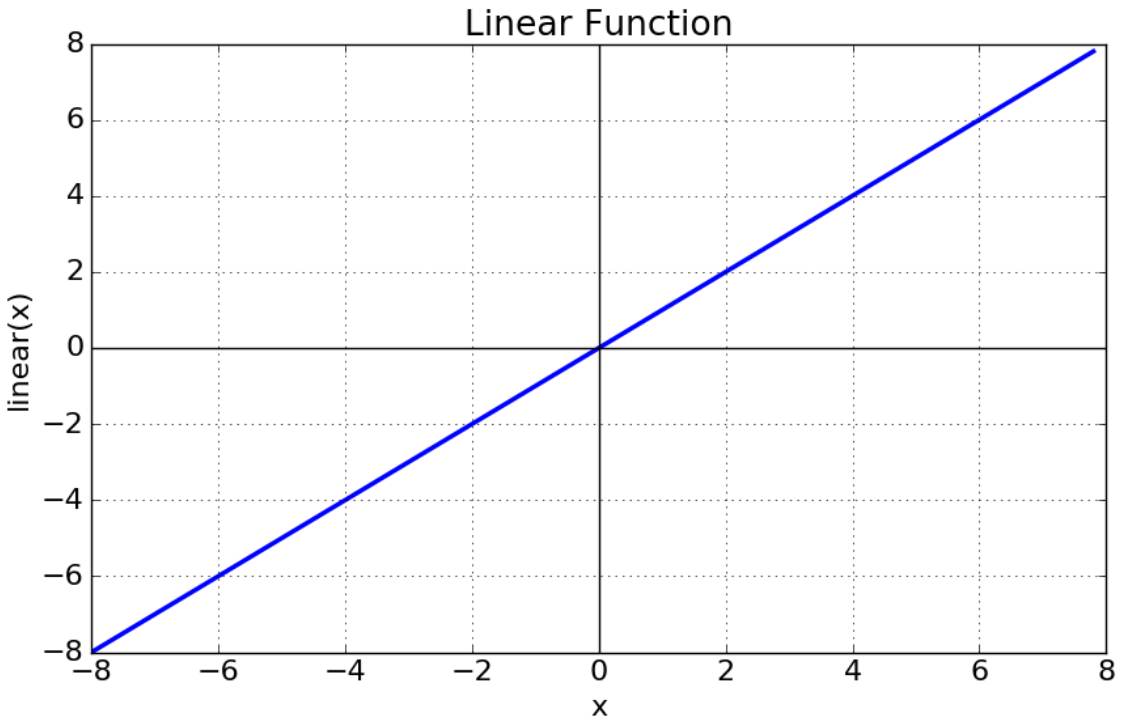

Figure 14. Identity Activation Function. *This function made the feed forward neural network less accurate due to collapsing the layers.*

Other parameters of the neural network had to be set for the multilayer perceptron regressor model, which include: the maximum number of iterations, the validation fraction, verbose, warm start, random state, early stopping and alpha. The neural network operates until it converges, determined by the tolerance and the maximum number of iterations. The validation fraction is set to 0.05 and represents the fraction of the training data which is set aside for early stopping [20]. Verbose was set to true in order to print progress messages in the command window. False warm start allows the model to erase the solution from the previous iteration and avoid using it as first iteration. Random bias initialization is set to none to avoid random numbers generation for weights and biases initialization. Early stopping is set to true and the model sets aside 10% of the training data to use it as validation and ends training when validation score is not improving by, at list, the specified tolerance. The alpha value of the feed forward neural network, also called L2 penalty (regularization term), is used in order to reduce the model from overfitting where the model could predict the output very accurately using the training data but, when using the testing set, the model would perform poorly. The alpha value of the neural network has been changed with respect to other parameters of the network in order to obtain optimal results. The training of the feed forward neural network is performed using the function implemented in the multilayer perceptron regressor model, fit (X , y). Where X represents the input data to the network. In this case, X is a matrix containing the different random energy group structures. The y variable contains the target values the neural network is trying to predict. In this case, y is an array containing the values of the neutron multiplication factors that generated the 20-group energy

[20] "Sklearn.Neural_network.MLPRegressor — Scikit-Learn 0.23.0 Documentation."

structures. These two variables contain all the data implemented in Python, without defining training and testing sets. The training set is defined when we use the function

predict (X_test). The matrix X_test contains the testing data that is being fed to the network, particularly 30% of the total data.

3.2 Data

In order to perform accurate simulations in machine learning, it is crucial to use as much data as possible. Depending on the number of samples used, the results will lead to more or less accurate results. "Training data is crucial in machine learning to memorize information for future predictions using the testing set" [21]. It is impossible to train any model without having any training data and predicting parameters of interest. Once the training of the machine learning tool is accomplished, it is crucial to test its performance. A small quantity compared to the training data is saved to test the model and check if it predicts the parameters of interest correctly. Another set of data is needed, the testing set or validation data.

For a nuclear reactor to be modeled correctly, many complex data are needed. First, the collision probability model needs user defined parameters to define the model we are testing. Some of these parameters include the geometry of the nuclear reactor's core, the number and type of nuclides used, or the moderator selected. Particularly, we are testing a fixed light water reactor containing only four nuclides (U235, U238, H1 and O16) and

[21] LLC, "Understanding the Importance Of Training Data In Machine Learning."

water as moderator. Then, we used ENDF data provided by ENDF/B-VII.1 Incident-Neutron Data. For each nuclide used in the model, ENDF data provides the continuous cross section plots for every kind of interaction inside the core of the nuclear reactor (absorption, elastic and gamma production interaction). These plots are showed in the previous section.

As stated in the above sections, the objective of the collision probability model is to discretize the continuous cross section and flux of the model using energy group structures. Consequently, we have used 18,034 random energy group structures to discretize the collapsed cross section plot and, as a result, the neutron flux spectrum.

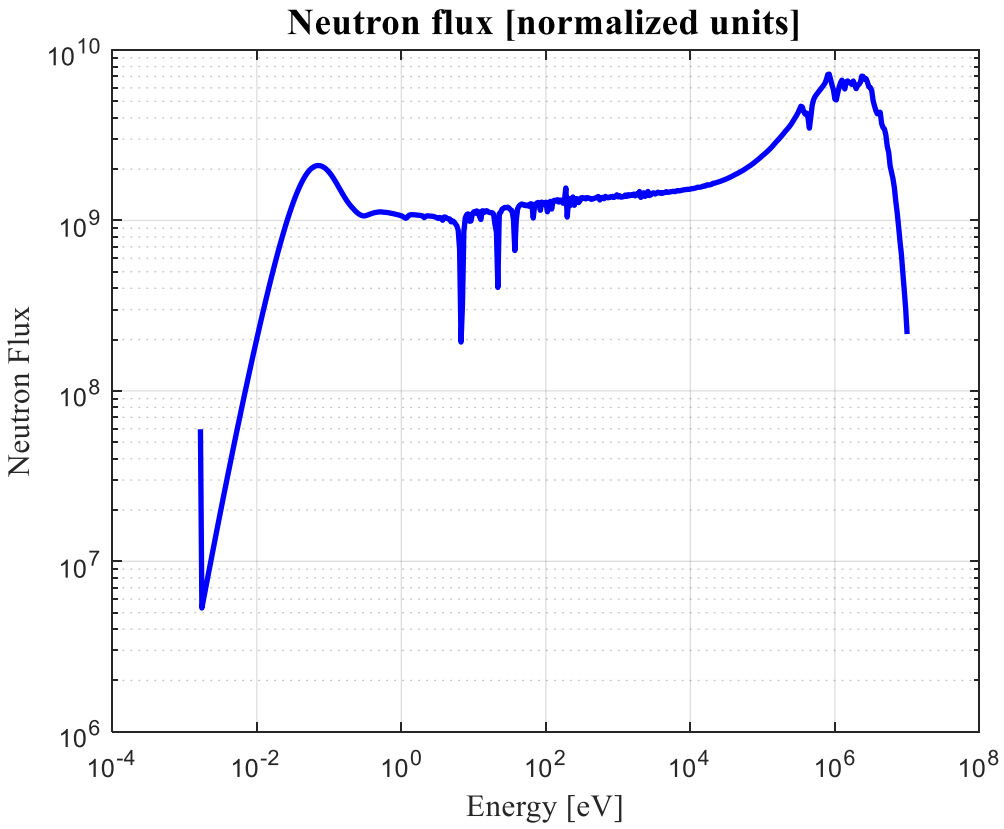

Figure 15. Neutron Flux Spectrum. *The flux spectrum is in normalized units, as its magnitude depends on the power level of the reactor.*

The neutron flux spectrum is used in normalized units in order to be useful for other nuclear reactor, as its magnitude depends on the power of the reactor. The energy of the incoming neutron is defined in logarithmic scale. In the flux spectrum, the neutrons are born in a Watt distribution with high energy. Afterwards, they are downscattered into the thermal region with low energy. In the middle of the neutron flux spectrum it can be observed that some neutrons are absorbed by resonances.

Each of the 18,034 random energy group structures is formed by 20 groups and the discretized collapsed cross section and neutron flux spectrum outputs an infinite neutron multiplication factor. Consequently, there are 18,034 neutron multiplication factors for each random energy group structure which are used as input to the feed forward neural network. Because the energy group structures are random and only formed by 20 groups, the related neutron multiplication factor can be either accurate or not. The accuracy of this factor will depend on how the group structures are randomly formed. As mentioned above, in order to obtain an accurate K_∞ the random group structure would need to have enough energy bins where the resonances of the cross section are not narrow enough for the narrow resonance approximation to hold. The target values that the network is trying to predict are the neutron multiplication factor and is set as an input to the neural network being an array of size 18,034 for each energy group structure. The other input to the network is the random 20-group structures, which is a matrix containing 18,034 rows and 20 columns. A training and testing set have been generated for the network to learn. Training is performed using 70% of the input data and validation of the neural network is done using the remaining data. The performance of the feed forward neural network was affected by the amount of testing and training data used.

3.3 Algorithms

The main focus of this project is the development of the feed forward neural network and its performance. How the algorithms were set up is explained in this section.

3.3.1 Initialization of the Feed Forward Neural Network

The feed forward artificial neural network is initialized using a previously defined function, initializeFFNN(). This function initializes the network given the parameters and hyperparameters stated in the "Specifications" section as input and returns the neural network model (gpr). To initialize the feed forward neural network it is necessary to previously import the multilayer perceptron regressor model provided by Scikit-Learn package.

```
from sklearn.neural_network import MLPRegressor

def initializeFFNN(solvertype, activationtype,HL_sz, vldF, maxIters, alpha_value):

    gpr = MLPRegressor(max_iter=maxIters,tol=1e-5,solver=solvertype,

            activation=activationtype,
```

validation_fraction=vldF,

learning_rate='adaptive',

hidden_layer_sizes=HL_sz,

verbose =True,

warm_start=False,

random_state=None,

early_stopping=True,

alpha=alpha_value)

return gpr

3.3.2 Training of the Feed Forward Neural Network

To train the artificial feed forward neural network there is a specific function that performs the task, doTraining(). This function takes the modeled artificial neural network (gpr), the predicted neutron multiplication factor of each 20-group structure from VBUDS3 (y), the total group density (X) and the testing group density (X_test) as inputs.

The function fits and predicts the data and returns the predicted neutron multiplication factor using the multilayer perceptron regressor.

```
def doTraining(gpr, X, y, X_test):
    trainingOutput=gpr.fit(X, y)
    yPred = gpr.predict(X_test)
    return trainingOutput, yPred
```

3.3.3 Utilities

A "Utilities" script was created with the purpose of processing all the data obtained from the collision probability code. In this script, several functions are defined to load the data, process it and feed it in the neural network. LoadData() brings the data generated by VBUDS3, in particular consists of a MATLAB file containing the random energy group structures used in the collision probability and their corresponding neutron multiplication factor. The groupStruct variable is a matrix of 18,034 rows and 21 columns. The energy axis is divided in 21 bins to obtain 20 groups as a result. The kinf variable is a 18,034-size array containing the neutron multiplication factor predicted by VBUDS3 using each random energy group structure. The function returns both parameters.

```
def LoadData(datapath):
    matContents = sio.loadmat(datapath)
    kinf = np.transpose(matContents['kinf']).ravel()#Flattens array
    groupStruct = np.transpose(matContents['groupStruct'])
    return kinf, groupStruct #returns contents of file
```

Once the data is loaded in Python, it needs to be processed. The data is processed to meet dimensions using ProcessData(). This function receives as input the path in which it was saved the mat file with the corresponding data of flux, neutron multiplication factor and group structures, the percentage of data used and the number of decades. The number of decades represents the number of parts in which the x axis of the neutron flux spectrum is divided, and it will be used to make group densities and generate an input array to the network. The defined function returns the training data and the k_∞ array. ProcessData() uses MakeGroupDensity() function to generate group densities and use them to feed the neural network.

```
def ProcessData(datapath, percOfData, nDecades):
    kinf, groupStruct = LoadData(datapath)
    Nsamples,Ngroups = groupStruct.shape
    NtrainingSamples = int(round(Nsamples*percOfData))
    X = groupStruct[:NtrainingSamples,:(Ngroups)-1]
```

y = np.zeros((1,NtrainingSamples))

y=kinf

trainingData=MakeGroupDensity(X,nDecades)

return trainingData, y

MakeGroupDensity() receives the matrix X with the random energy group structures and the number of bins in which the energy axis will be divided (nDecades). The upper and lower limit in which the bins will be generated is user defined. In this case, between -3 and 7. This is because the energy axis will be transformed in logarithmic scale and the energy upper and lower levels are 10^{-3} and 10^{7} eV. The number of decades affects the performance of the neural network, stablishing in the project a total of 200 bins. The matrix X is transformed to logarithmic scale generating, consequently, x_lognorm. Once the energy domain is divided in 200 equally logarithmically spaced bins, each bin is filled with an integer number corresponding to the number of energy boundaries out of the original group structure that lie within the given energy interval. A matrix of densities is created and returned using decadeDensity. This matrix will be used as the input neuron to the neural network.

def MakeGroupDensity(X, nDecades):

decades = np.linspace(-3, 7, nDecades)

x_lognorm = np.log10(X)

decadeDensity=np.zeros((x_lognorm.shape[0], len(decades)-1))

```
for N in range(0,x_lognorm.shape[0]):
    for i in range(0,len(decades)-2):
        for j in range(0,len(x_lognorm[N,:])):
            if x_lognorm[N,j] >= decades[i] and x_lognorm[N,j] < decades[i+1]:
                decadeDensity[N,i]=decadeDensity[N,i]+1
return decadeDensity
```

The main script calls the function makeFractions(), which takes as input the number of samples, the validation factor the test fraction, the decade densities matrix and the neutron multiplication factor array. The number of samples correspond to the whole amount of data (18,034 random energy group structures). The validation factor is 0.05, explained in the initialization of the feed forward neural network. The test fraction can be defined and modified by the user and represents the percentage of testing data that will use the neural network. I have tested the network with 30% of the total data. Consequently, there are a total of 12,624 random energy group structures used for training the feed forward neural network. The decade densities matrix and the neutron multiplication factor array are modified to obtain a new matrix and array for training and testing. The function returns the training and testing variables, as well as the new validation factor of the neural network.

```
def makeFractions(Nsamples, vldF, testF, trainingData, trainingAnswers):
    NtrainingSamples = int(round(Nsamples*(1 - testF)))
```

```
vldF_corr=vldF*Nsamples/NtrainingSamples

X = trainingData[:NtrainingSamples,:]

y = trainingAnswers[:NtrainingSamples].T

X_test = trainingData[NtrainingSamples+1:,:] # test data

y_test = trainingAnswers[NtrainingSamples+1:].T

return X, X_test, y, y_test ,vldF_corr
```

Once the feed forward neural network is initialized and trained, I defined a function to determine the accuracy of the network, numRightReg(). This function uses as inputs the predicted neuron multiplication factors from the neural network and the ones used for testing (only 30% of the total data). The defined function uses a validation factor to compare whether the predicted k_∞ can be considered "equal" to the k_∞ obtained from the collision probability code. This validation factor is set to 0.95, meaning that the prediction from the network is considered correct if both values only differ by 5%. The function defines a list where it can be seen using its index in which position are nicely predicted neutron multiplication factors. It returns the number of k_∞ values which were inside the specified tolerance of 5% and those which were not, as well as the index in the list where both can be found.

```
def numRightReg(yPred, y_test):
    Validation_Score=0.95
    countRight=0
    countWrong=0
    indexRight=[]
    indexWrong=[]
    for i in range(0,len(y_test)):
        if 1-((y_test[i]-yPred[i])/y_test[i])>=Validation_Score:
            indexRight.append(i)

            countRight=countRight+1
        else:
            indexWrong.append(i)
            countWrong=countWrong+1

    return countRight, countWrong, indexRight, indexWrong
```

In order to know the error distribution and the performance of the feed forward neural network, an error histogram is plotted using ErrorHistogram(). The function compares the predicted array of neutron multiplication factors and the ones used for testing, which are the inputs of the function. The error histogram plot will be analyzed under the "Results" chapter.

```
def ErrorHistogram(yPred, y_test):
    Error=(yPred-y_test)/yPred
    plt.hist(Error, bins = 20)
    plt.show()
    return
```

3.3.4 Main

The main script of the feed forward neural network uses "Utilities" and "FFNN" to run the network and obtain the results. The libraries and sub-scripts are imported from the Scikit-Learn package and used throughout the script. The first thing done by the program is load and process the data from the collision probability code, in format of a MATLAB file. Some user defined parameters are created such as the validation fraction, the test fraction, the maximum number of iterations until convergence, the regularization term, the number of hidden layers and neurons in each hidden layer, the data fraction, the solver

type used by the network and the activation type. These parameters are defined by the user, but I have used a systematic optimization technique to learn which of the hyperparameters of the network produced the best results. This optimization will be explained in the "Result's" section. Them the feed forward neural network is initialized and trained, determining the number of neutron multiplication factors which were predicted correctly. Once this value is known, the accuracy of the network can be computed given the previously user defined parameters. The performance of the neural network depends on many parameters. Some of them are learned by the network, but the hyperparameters have to be defined by the user. Finally, the model plots a histogram to show the behavior of the neural network and how many neutron multiplication factors were predicted correctly.

```
from __future__ import unicode_literals, print_function, division
import Utilities
import FFNN
import numpy as np

datapath = '/Users/Alvaro/Desktop/ICAI/TFG/Jessica/Artificial Neural Network/AllCurrentLibraries2.mat'
print('Loading In Data')
allData, y_direct = Utilities.ProcessData(datapath, 1,200)
print('Finished Loading Data')
Nsamples,Ndecades = allData.shape
vldF=0.05
```

testF=0.3

maxIters=50000

alpha=0.000000001

HL_sz=(24,24)

dataFrac=1

X, X_test, y, y_test, vldF_corr = Utilities.makeFractions(Nsamples, vldF, testF, allData, y_direct)

solvertype='lbfgs'

activationtype='relu'

gpr = FFNN.initializeFFNN(solvertype, activationtype,HL_sz, vldF_corr, maxIters, alpha)

trainingOutput, yPred= FFNN.doTraining(gpr, X, y, X_test)

countRight, countWrong,indexRight,indexWrong = Utilities.numRightReg(yPred, y_test)

print('Of the %d samples tested'% len(y_test))

print('The forward fed neural net predicted:')

print('%i right'% countRight)

print('%i wrong'% countWrong)

print(str(100(countRight)/(countWrong+countRight)),"% Accuracy")*

Accuracy=100(countRight)/(countWrong+countRight)*

Utilities.ErrorHistogram(yPred, y_test)

3.3.5 Simulated Annealing

Hyperparameter optimization techniques have been developed in order for the artificial neural network to predict parameters which cannot be learned by the network itself. In this case, it has been implemented the simulated annealing optimization strategy. "Simulated annealing is metaheuristic technique to approximate the global optimum of a certain objective function". In some cases, it is preferable to use simulated annealing rather than gradient descent when the search space is discrete and when the search space is large. "Simulated annealing strategy comes from the metallurgic term "annealing". In metallurgy, the material is heated and cooled down controllably to increase the size of the crystals and reduce the number in defects in the part". The analogy of slow cooling in simulated annealing refers to the slow decrease in the probability of accepting worse solutions as all the possibilities are explored. "The temperature decreases in each step starting with a value and targeting zero"." To converge to optimal solution, the algorithm randomly selects a solution close to the previous one and measures the quality, leading to temperature decrease probabilities in each step" [22]. In this project, simulated annealing was implemented in the model to optimize the feed forward neural network and obtain a better accuracy. The function defined in "Utilities" to model the simulated annealing algorithm is below.

[22] "Simulated Annealing."

```
def OptimizeHyperparameters(X, X_test, y, y_test,gpr):
    hl_sizes=[]
    for i in range (1,65,1):
        for j in range (1,65,1):
            for l in range (1,65,1):
                hl_sizes.append((i,j,l))

    for k in range (1,65,1):
        hl_sizes.append(k)

    neural_net_hyperparams = {'hidden_layer_sizes':hl_sizes,
                              'Learning_Rate':np.arange(0.001, 1.001, 0.001),
                              'alpha':np.arange(0.00001, 21.00001, 0.00001)
                              }

    sa = SimulatedAnnealGonzalo(gpr, neural_net_hyperparams, T=10.0, T_min=0.001,
alpha=0.75,
                                verbose=True, max_iter=1, n_trans=5, max_runtime=300,
```

```
        cv=3, scoring='max_error', refit=True)

    sa.fit(X, y)

    print(sa.best_score_, sa.best_params_)

    optimized_gpr = sa.best_estimator_

    y_test_pred = optimized_gpr.predict(X_test)

    return sa.best_score_, sa.best_params_
```

In this function, a grid of hyperparameters is first created for the simulated algorithm to search over. In this project, the hyperparameters which were analyzed were the regularization term (alpha), the number of hidden layers in the neural network and the number of neurons in each layer. The modeled artificial neural network using the multilayer perceptron regressor is used as input to the simulated annealing algorithm which searches over the stablished hyperparameters. The best hyperparameters are returned and used again with a modified MLPRegressor using the new proposed hyperparameters by simulated annealing.

```
Best_Score, Best_Params=Utilities.OptimizeHyperparameters(X, X_test, y, y_test,gpr)

gpr_best = FFNN.initializeFFNN(solvertype, activationtype,Best_Params['hidden_layer_sizes'], vldF_corr, maxIters, Best_Params['alpha'])

trainingOutput, yPred= FFNN.doTraining(gpr_best, X, y, X_test)
```

countRight, countWrong,indexRight,indexWrong = Utilities.numRightReg(yPred, y_test)

print('Of the %d samples tested'% len(y_test))

print('The forward fed neural net predicted:')

print('%i right'% countRight)

print('%i wrong'% countWrong)
print(str(100(countRight)/(countWrong+countRight)),"% Optimized Accuracy")*

Accuracy_best=100(countRight)/(countWrong+countRight)*

However, I have experienced some problems with simulated annealing trying to optimize the artificial neural network. Usually, the algorithm gets stuck in local maximum and does not reach global optimal hyperparameters to achieve best accuracy. But simulated annealing approach was useful to find a multilayer perceptron regressor model which lacks overfitting.

3.3.6 GridSearchCV

Many of the times failing to output hyperparameters which lead to better results due to not reaching global maximum, a new optimization method has been implemented." Grid search cross validation algorithm performs an exhaustive search of a specified grid of hyperparameters for a given estimator" [23]. This hyperparameter tuning technique has shown to be extremely powerful, as it can provide the best model solving the neural network for every combination of the hyperparameter's candidates grid. However, it is obvious that the computational effort is greater than simulated annealing. Also, the running of the artificial neural network is rather slow because the multilayer perceptron regressor has to be solved for every combination set in the hyperparameter's grid. But this issue improves significantly using the right type of solver for the network. In this project, it has been experienced that 'adam' provides the output neutron multiplication factor quicker than 'sgd' or 'lbfgs'. However, it is not completely solved if we want to search over a large grid of hyperparameters.

The grid search cross validation algorithm is implemented defining the following function. It uses the density training matrix, the neutron multiplication factor array and the multilayer perceptron regressor model to study as inputs. Stablishing a user defined grid of parameters, the function searches over every combination of hyperparameters and returns the list of optimal parameters along with the best score.

[23] "Sklearn.Model_selection.GridSearchCV — Scikit-Learn 0.23.1 Documentation."

```
def Random_search(X, X_test, y, y_test, gpr):
    alpha=np.array([0.00000000001, 0.0000000001, 0.000000001, 0.00000001, 0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1])
    hidden_layers=[]
    for i in range (1,65,1):

        for j in range (1,65,1):
            hidden_layers.append((i,j))
    param_grid={'hidden_layer_sizes':hidden_layers, 'alpha':alpha}
    random=RandomizedSearchCV(estimator=gpr, param_distributions=param_grid)
    random.fit(X,y)
    Best_params=random.best_params_
    Best_score=random.best_score_
    return Best_score, Best_params
```

The optimized hyperparameters are returned to the main script and input in a new multilayer perceptron regressor model which outputs the optimized accuracy.

Best_Score_GS, Best_Params_GS=Utilities.Grid_search(X, X_test, y, y_test,gpr)

gpr_best_GS = FFNN.initializeFFNN(solvertype, activationtype,Best_Params_GS['hidden_layer_sizes'], vldF_corr, maxIters, Best_Params_GS['alpha'])

trainingOutput, yPred= FFNN.doTraining(gpr_best_GS, X, y, X_test)

countRight, countWrong,indexRight,indexWrong = Utilities.numRightReg(yPred, y_test)

print('Of the %d samples tested'% len(y_test))

print('The forward fed neural net predicted:')

print('%i right'% countRight)

print('%i wrong'% countWrong)

print(str(100(countRight)/(countWrong+countRight)),"% Optimized Accuracy")*

Accuracy_best_GS=100(countRight)/(countWrong+countRight)*

Although performing a grid search cross validation is computationally expensive, doing a clever search and selecting not a very large range of hyperparameters can lead us to obtain the optimal accuracy very fast.

3.3.7 RandomizedSearchCV

Due to tough computational effort selecting a wide grid of hyperparameters and solving the model using GridSearchCV, it has been implemented in this project a random search cross validation method.

"While in grid search approach the user selects a grid of hyperparameters and the model is trained for every combination (which can be very inefficient if the grid is large), in random search cross validation approach the model selects random combinations of the user defined grid"." It is possible that random search will not find an accurate result as grid search, but the computational price is much lower" [24]. The main advantage of random search cross validation over grid search is the running time, and the first approach is more suitable when you need to explore larges grids of hyperparameters. The function is defined below, and it has been tuned to search over a different grid every time.

```
def Random_search(X, X_test, y, y_test, gpr):
    alpha=np.array([0.00000000001, 0.0000000001, 0.000000001, 0.00000001, 0.0000001, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1])
    hidden_layers=[]
    for i in range (1,65,1):
        for j in range (1,65,1):
```

[24] Worcester, "A Comparison of Grid Search and Randomized Search Using Scikit Learn."

hidden_layers.append((i,j))

param_grid={'hidden_layer_sizes':hidden_layers, 'alpha':alpha}

random=RandomizedSearchCV(estimator=gpr, param_distributions=param_grid)

random.fit(X,y)

*Best_params=random.best_params_*

*Best_score=random.best_score_*

return Best_score, Best_params

The RandomizedSearchCV model implemented by Scikit-Learn performs each simulation using the defined feed forward artificial neural network (gpr) and returns the best hyperparameters found to the main script, which runs the simulation again and computes the optimized artificial neural network accuracy. Once again, it can happen that the optimized accuracy is lower than the computed accuracy by the user defined hyperparameters. This happens because RandomizedSearchCV looks through the hyperparameter combinations randomly and not all the possibilities are tried out. The artificial neural network is trained using the obtained hyperparameters as follows.

Best_Score_RS, Best_Params_RS=Utilities.Random_search(X, X_test, y, y_test,gpr)

gpr_best_RS = FFNN.initializeFFNN(solvertype, activationtype,Best_Params_RS['hidden_layer_sizes'], vldF_corr, maxIters, Best_Params_RS['alpha'])

```
trainingOutput, yPred= FFNN.doTraining(gpr_best_RS, X, y, X_test)

countRight, countWrong,indexRight,indexWrong = Utilities.numRightReg(yPred, y_test)

print('Of the %d samples tested'% len(y_test))

print('The forward fed neural net predicted:')

print('%i right'% countRight)

print('%i wrong'% countWrong)

print(str(100*(countRight)/(countWrong+countRight)),"% Optimized Accuracy")

Accuracy_best_RS=100*(countRight)/(countWrong+countRight)
```

Chapter 4. **Results**

In this chapter it is described the analysis of the results in the base case in section 4.1 and the analysis of relevant configurations of the artificial neural network in section 4.2 using trial and error method. Also, in sections 4.3 and 4.4 the optimal artificial neural network structure is described using grid search and random search cross validation respectively.

4.1 Results of the base case

The collision probability code predicts the infinite neutron multiplication factor using a discrete energy group structures formed by 500 groups. That way, the discretized neutron flux spectrum provides an accurate k_∞ output. This neutron multiplication factor will be the benchmark for reaching accuracy regarding the random 20-group energy structures. Using 500 groups will allow VBUDS3 to generate a discrete neutron flux spectrum and produce a neutron multiplication factor accurate enough. The obtained value of k_∞ is 1.3815. The histogram of values predicted by the collision probability code using 20 group random energy structures is the following.

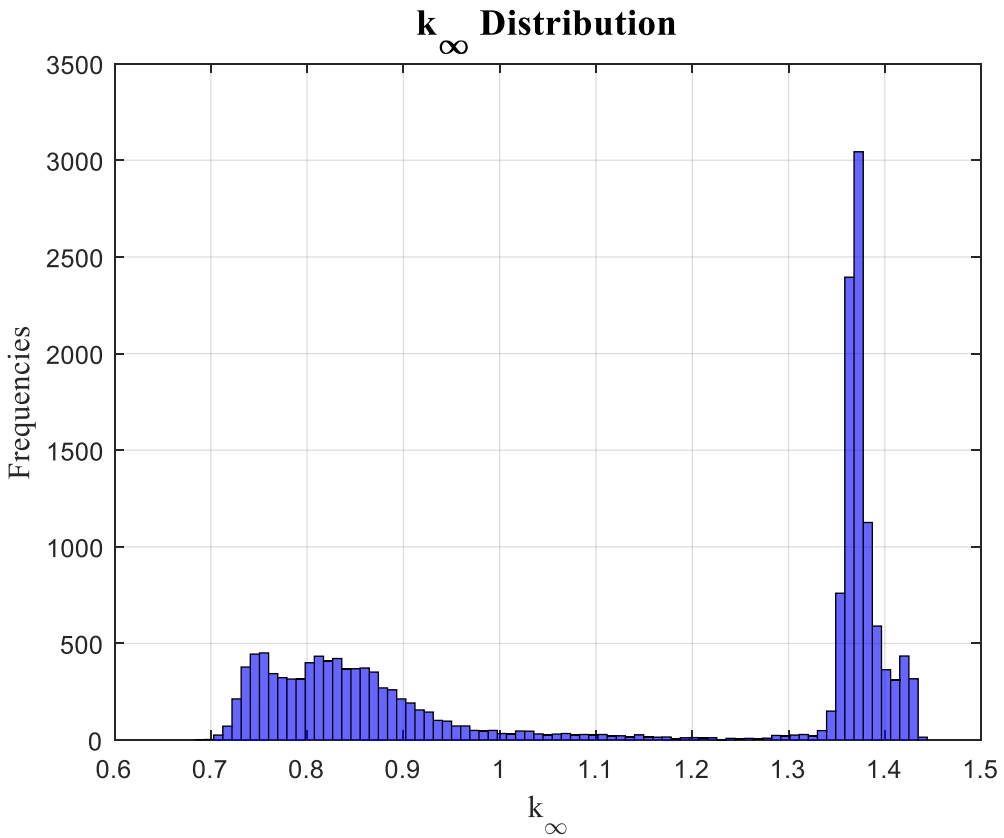

Figure 16. Neutron Multiplication Factor Distribution. *The distribution is graphed using random energy group structures of 20 groups.*

The target k_∞ value is 1.3815 and, although there are some configuration structures which are not close to that value, there is a large percentage of 20-group energy structures which are close to the target value. The members of the training set were initially classified by comparing their corresponding k_∞ values with the benchmark.

Members with k_∞ within $\Delta k_\infty = 2.91\%$ of the benchmark were classified as having good performance. The Δk_∞ value was chosen such that there was an even split between good and poor performing group structures in the training data. It can be observed in the chart that most of the predicted k_∞ values are between 1.35 and 1.45 which means that, with the provided energy group structures, the core of the nuclear reactor would be generating more neutrons with time and would be in a "supercritical" condition. When we use few group structures, it is difficult to find the right boundaries which predict an accurate neutron multiplication factor, as shown in the left side of the chart. The results obtained in this side of the chart mean that the random energy group structures could not discretized very well the cross section and the neutron flux spectra, leading to inaccurate results of the neutron multiplication factor.

The accuracy of the feed forward neural network is based on the input parameters. However, as mentioned in previous sections, there are some parameters which can be learned by the network but others which must be user defined, the hyperparameters. There have been developed certain optimization techniques for selecting the appropriate hyperparameters but first it will be analyzed the accuracy of the network using defined hyperparameters by the user. The main interesting hyperparameters regarding the multilayer perceptron regressor model which will this project focus are the number of neurons in each hidden layer, the solver type used in the network and the regularization term parameter. For simplicity, it will be assumed that the number of hidden layers in the model are two and the activation type is 'tanh' as it gave the best results of accuracy. But

the neural network has been modeled using random user defined parameters to test its accuracy. Also, a reasonable number of neurons for each layer is in the range from 1 to 64. The first structures of the network have been modeled assuming the same number of neurons in each hidden layer. It has been taken into account the execution time of the feed forward neural network and the results for each solver type are shown in the following diagram.

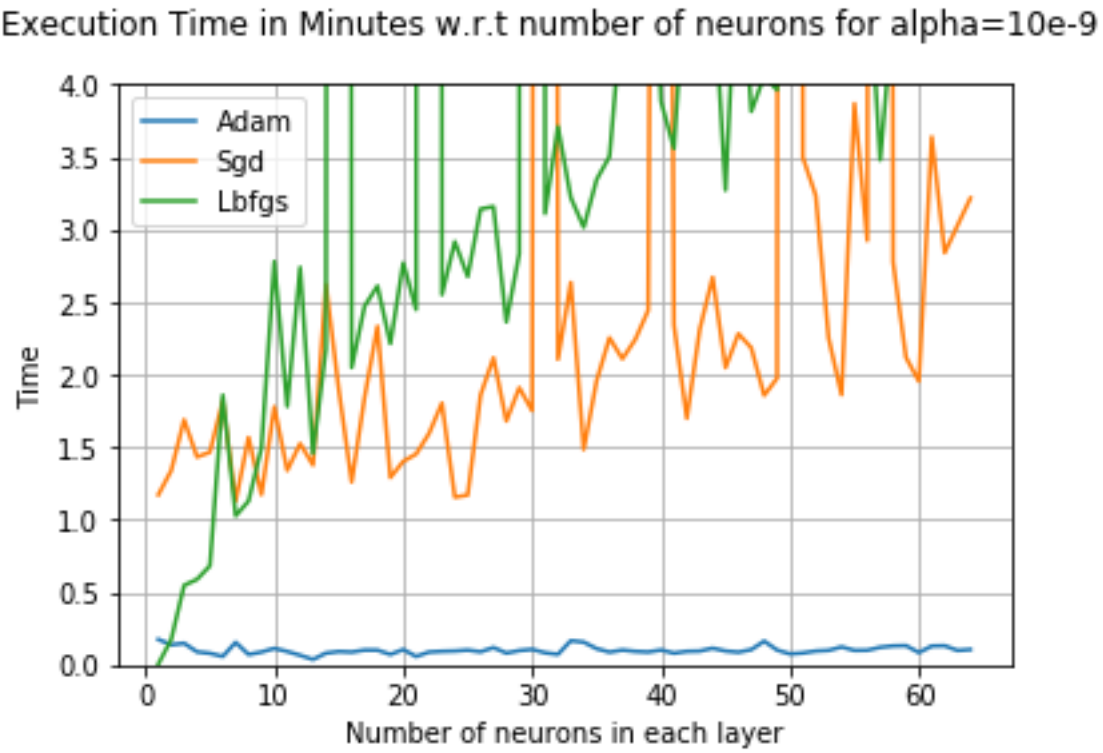

Figure 17. Execution Time in Minutes with Respect to the Number of Neurons in Each Hidden Layer. *The analysis of this model assumes that there is the same number of neurons in each hidden layer and the regularization term equals* 10^{-9}.

The plot shows clearly the efficiency of each solver. 'Adam' solver proves to be the most time efficient dealing with the amount of data fed into the network. In contrast, 'lbfgs' solver uses even hours to compute one iteration. There is also a tendency in stochastic gradient descent and 'lbfgs' solvers to increase computational exacution time as the number of neurons increases, whereas 'adam' proves to be ore stable over the amount of nodes.

At first, the artificial neural network is initialized with a random regularization term parameter and solver type using different configurations for the number of neurons in each hidden layer. This approach was set in order to know which parameters performed better. Alpha was set to 3.75 and the first solver type chosen was 'adam' as the running time is low." 'Adam' is the default solver provided by the multilayer perceptron regressor model and is based on a stochastic gradient descent optimizer developed by Kingma, Diederik and Jimmy Ba" [25]. Also, 'relu' activation neurons were first used in order to check the network's performance. The results of accuracy with respect to number of neurons in each layer using 'adam' solver are the following.

[25] "Sklearn.Neural_network.MLPRegressor — Scikit-Learn 0.23.0 Documentation."

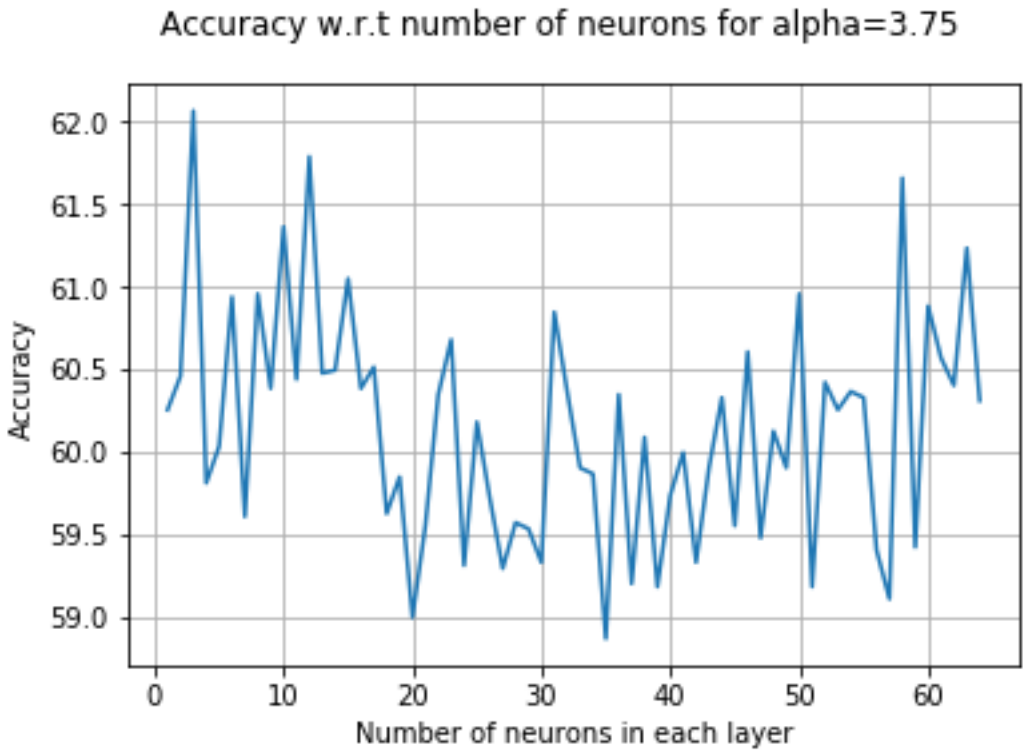

Figure 18. Accuracy of the Neural Network with Respect to the Number of Neurons in Each Layer Using 'adam' Solver and 'Relu' Activation. *The model used a regularization term of 3.75.*

As it can be observed in the chart, with the regularization term equal to 3.75 and using 'adam' solver, the accuracy of the network does not follow a recognizable path. Instead, this model assimilates to a sine graph being the highest accuracy near to 62% which is not very high. New solver types have been simulated using the multilayer perceptron regressor to check for improvement in accuracy.

4.2 Relevant Configurations

In order to improve accuracy of the artificial neural network, there have been developed certain relevant models making some modifications.‘Sgd’ refers to stochastic gradient descent and, performing the simulation with the same regularization term, the results of number of neurons in each hidden layer with respect to accuracy of the neural network are the following.

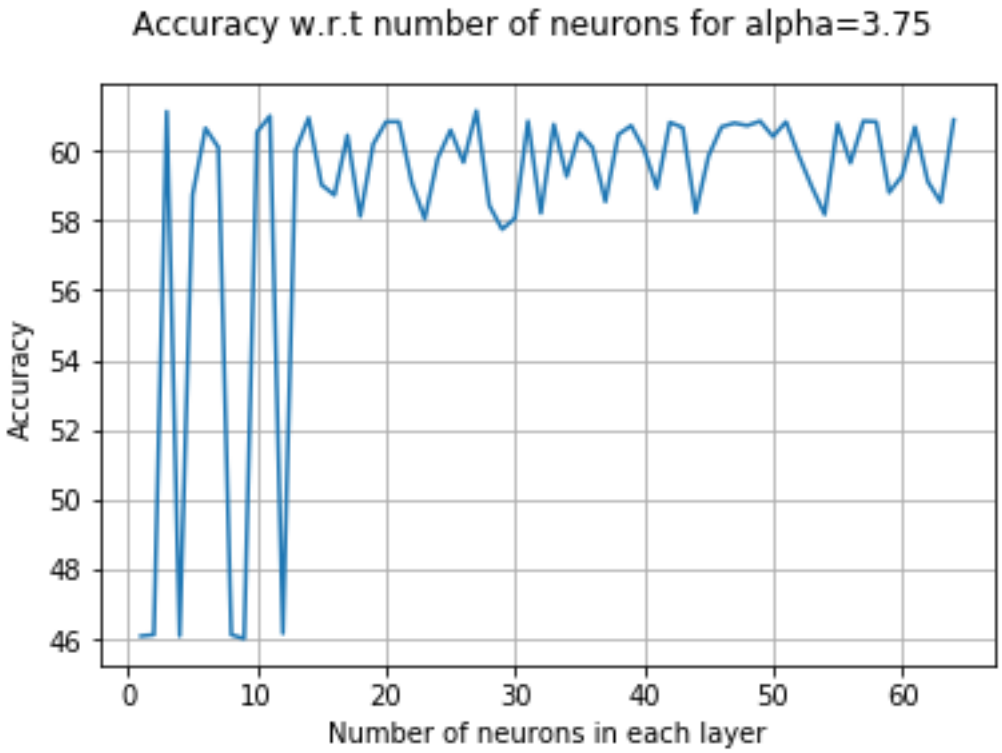

Figure 19. Accuracy of the Neural Network with Respect to the Number of Neurons in Each Layer Using 'sgd' Solver Using 'Relu' Activation. *The model used a regularization term of 3.75.*

This model starts with a sine shape of accuracy with respect to number of neurons in each layer and then stabilizes in a range of values between 58% and 61% of accuracy. Although this model makes the neural network more stable after the value of 11 neurons in each hidden layer, it is suitable to find a better and more stable model. Also, the best accuracy of this model is still less than the best accuracy of the previous model.

'Lbfgs' solver type is an optimizer in the family of quasi-Newton methods, as mentioned in previous sections. Using this solver with the same regularization parameter we obtain the below results.

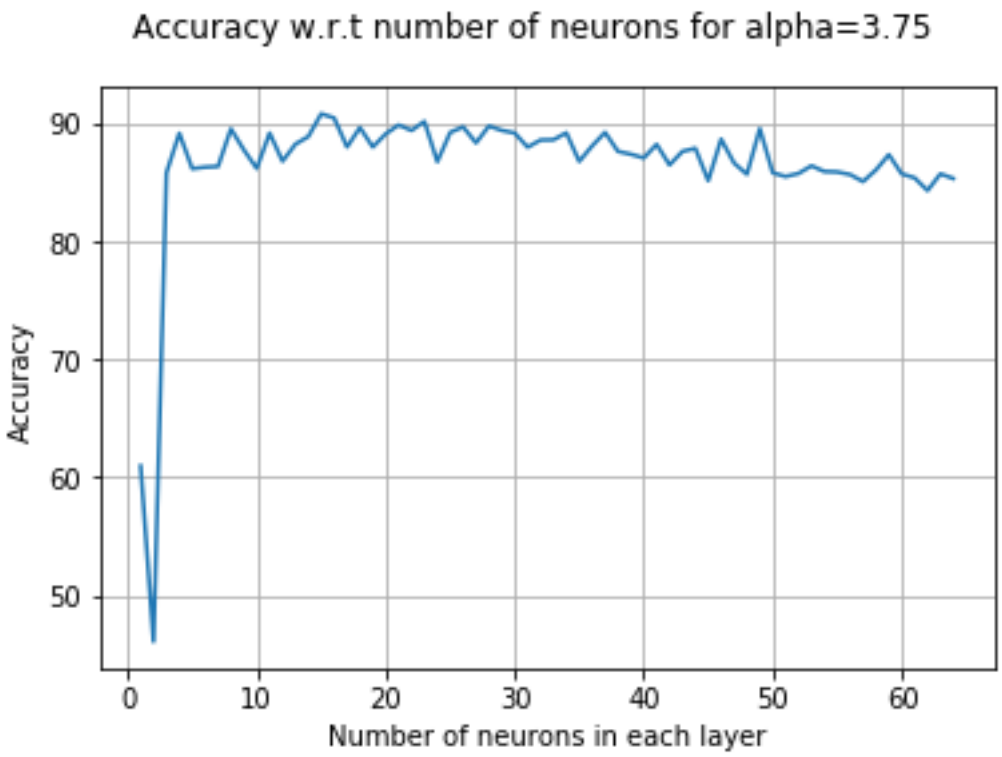

Figure 20. Accuracy of the Neural Network with Respect to the Number of Neurons in Each Layer Using 'lbfgs' Solver Using 'Relu' Activation. *The model used a regularization term of 3.75.*

The main difference with respect the previous two models is the improvement of accuracy of the artificial neural network. Now, the best value of accuracy is above 90%, which is significant in artificial neural networks. This peak is reached with 15 neurons in each hidden layer but, as it can be seen in the graph, there are small fluctuations of accuracy leading to a reduction tendency when the number of neurons exceed 50. It is not until neuron 5 where the neural network achieves reasonable accuracy with this model.

The low values of accuracy of the network lead us to modify the regularization term, realizing that there is significant improvement when selecting alpha equal to 10^{-9} .

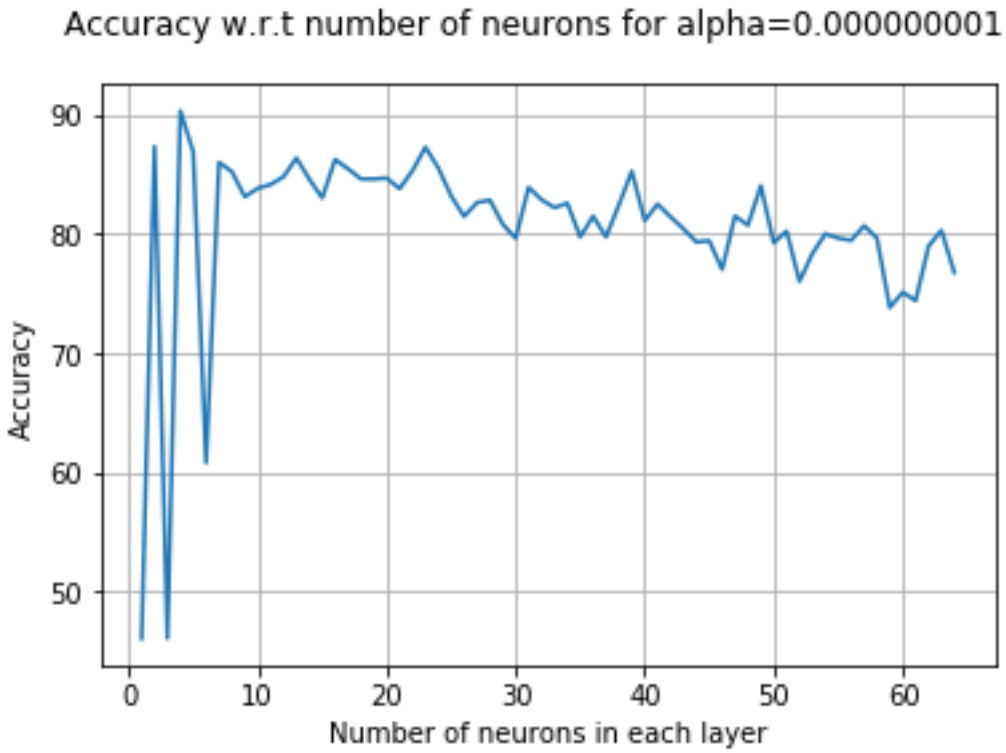

Figure 21. Accuracy of the Neural Network with Respect to the Number of Neurons in Each Layer Using 'adam' solver. *The model used a regularization term of* 10^{-9}.

Modifying the regularization term significantly improves the accuracy of the 'adam' solver, reaching an accuracy of more than 90% with 4 neurons in each hidden layer. However, it can be observed in the graph that there are small fluctuations when the number of neurons overpass 8 having a decreasing tendency in accuracy.

The results obtained using 'sgd' solver with this model are presented in the below chart.

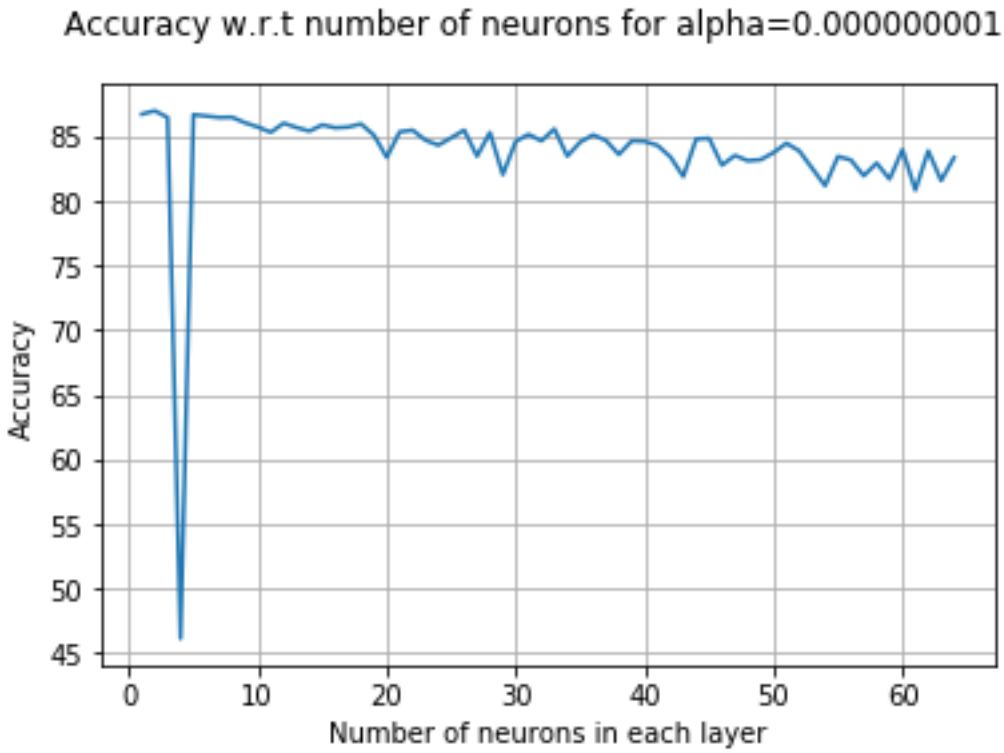

Figure 22. Accuracy of the Neural Network with Respect to the Number of Neurons in Each Layer Using 'sgd' solver. *The model used a regularization term of* 10^{-9}.

The 'sgd' system is even more stable than the one before. Also, the range of fluctuation is less, and the best accuracy reaches a higher level than the previous 'sgd' model. This accuracy is reached with only one neuron in each hidden layer being 86.67%. There is still some tendency of reducing the accuracy as the number of neurons increases, but in a much more relaxed way than 'adam'.

As mentioned in previous sections, 'lbfgs' solver leads to best results regarding this multilayer perceptron model. In artificial neural networks, it is suitable to search for the parameters which lead to best results. In our case, 'lbfgs' solver reaches the highest accuracy being the results shown in the following graph.

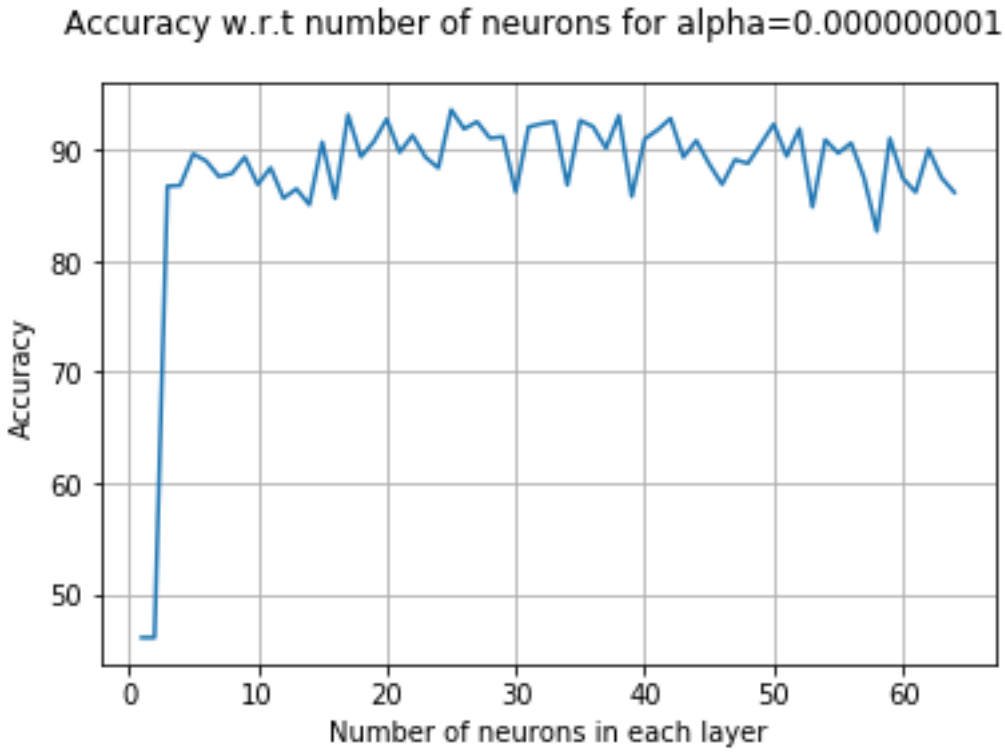

Figure 23. Accuracy of the Neural Network with Respect to the Number of Neurons in Each Layer Using 'lbfgs' solver. *The model used a regularization term of* 10^{-9}.

As we can see in the graph shown above, the artificial neural network stabilizes earlier than the past models using ‘adam’ or ‘sgd’ solver types. This model of the network presents the most accurate results obtained initializing the regularization term and the activation type randomly (93.6033% and 24 neurons in each of the two hidden layers). It has also been observed that the neural network performs accurately using half of the data using ‘lbfgs’ solver 10^{-9} as regularization term and 24 neurons in each of the two hidden layers.

The method used above to seek good accuracy of the feed forward neural network is based on hit and error at first. The graphs showing the number of neurons in each hidden layer and the output accuracy of the network is, computationally speaking, not very tough because there are the same number of neurons in each hidden layer. These plots provide us with an idea of certain hyperparameters which can lead to reasonable accuracy. The optimal results found for each solver and maintaining the regularization term fixed are shown in tables below.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**Adam**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**ReLU**
Neurons in each Hidden Layer	**(4,4)**
Accuracy	**90.50%**

Table 2. Feed Forward Neural Network Hyperparameters Using 'adam' Solver with Its Corresponding Accuracy and 'Relu' Activation.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**sgd**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**ReLU**
Neurons in each Hidden Layer	**(1,1)**
Accuracy	**86.67%**

Table 3. Feed Forward Neural Network Hyperparameters Using 'sgd' Solver with Its Corresponding Accuracy and Relu Activation.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**lbfgs**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**ReLU**
Neurons in each Hidden Layer	**(24,24)**
Accuracy	**93.60%**

Table 4. Feed Forward Neural Network Hyperparameters Using 'lbfgs' Solver with Its Corresponding Accuracy and Using 'Relu' Activation.

It can be observed in the previous tables that the optimal result where the neural network predicted more accurately the neutron multiplication factor is where the model uses 'lbfgs' solver, the regularization term is 10^{-9} and the number of neurons in each hidden layer is 24, resulting in 93.60% accuracy.

I have tested the artificial feed forward neural network using the possible activation functions, 'lbfgs' solver and regularization term equal to 10^{-9}, as it provided accurate results using 'relu', and the comparison looks as follows. The best configuration was found to be using 'tanh' activation function and 13 neurons in each hidden layer. Again, for simplicity, it has been analyzed the model using only two hidden layers and the same number of neurons in each hidden layer.

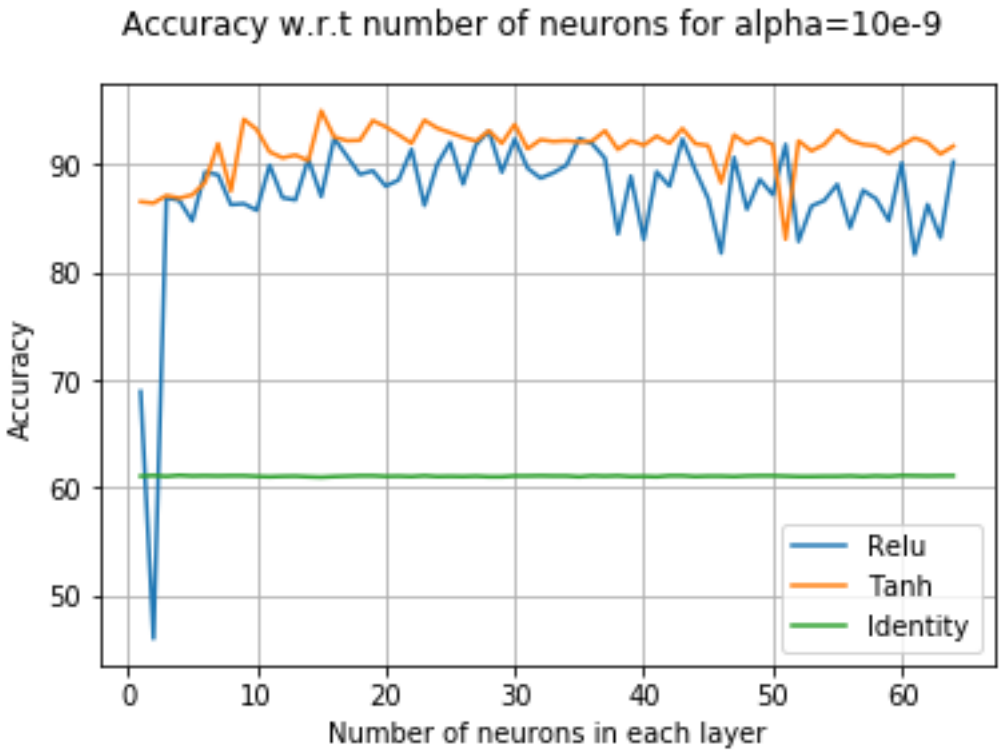

Figure 24. Accuracy with Respect to Number of Neurons in Each Hidden Layer Using the Different Activation Functions and 'Lbfgs' Solver. *Tanh activation function provides the most accurate model.*

This chart helps to realize the power of 'tanh' activation function. Almost in every case, the accuracy of the model is above 'relu' and 'identity' activations. The neural network with 13 neurons in each hidden layer provides the best accuracy using 'lbfgs'. Supposing that the architecture of the network which uses 13 neurons in each hidden layer and a L2 term equal to 10^{-9} provides accurate results using the other solver types, their accuracy has been compared. The comparison with 'adam' and 'sgd' solvers is presented below.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**Adam**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**Tanh**
Neurons in each Hidden Layer	**(13,13)**
Accuracy	**86.00%**

Table 5. Accurate Model Using Tanh Activation and 'Adam' Solver.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**Sgd**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**Tanh**
Neurons in each Hidden Layer	**(13,13)**
Accuracy	**84.91%**

Table 6. Accurate Model Using Tanh Activation and 'Sgd' Solver.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**lbfgs**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**Tanh**
Neurons in each Hidden Layer	**(13,13)**
Accuracy	**94.90%**

Table 7. Accurate Model Using Tanh Activation and 'Lbfgs' Solver.

Using trial and error approach it has been found that the best model of the network consists in two hidden layers with 14 neurons in each hidden layer, a regularization term of 10^{-9}, 'lbfgs' solver type and 'tanh' activation, giving a value of 94.90%

4.3 GridSearchCV Results

Grid search approach was used in the feed forward artificial neural network to optimize the number hidden layers, the number of neurons in each hidden layer and the regularization term. The computational effort by the model depends on the solver type used. 'Adam' is the fastest solver and performing a grid search over some interesting

hyperparameters can be fairly quick. Although the experience states that if 'lbfgs' is the slowest solver for this particular model, it was worth searching over a certain grid as this solver provides the best accuracy, but no models provided more accuracy than 94.90%. The use of 'sgd' solver is not very practical, as the accuracy obtained is not as high as using 'lbfgs' and the model converges slowly.

Due to computational effort, grid search cross validation approach was performed using 'adam' solver hoping that the selected grid contained optimal hyperparameters. The first user defined grid consisted of searching from 1 neuron in each hidden layer to 10 neurons using a regularization term of 10^{-9} as it provided the best accuracy range using this solver as shown in the graph above. The optimal results found by the network are summarized in table below.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**adam**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**ReLU**
Neurons in each Hidden Layer	**(7,7)**
Accuracy	**84.32%**

Table 8. Feed Forward Neural Network Hyperparameters Using GridSearchCV Approach, 'adam' Solver and 'Relu' Activation with Its Corresponding Accuracy.

Searching for optimal results, it was found that the model produced better results using 0.1 as regularization term. A wider grid of hidden layers using this hyperparameter was developed, obtaining the following results.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**adam**
L2 regularization parameter	**0.1**
Activation function	**ReLU**
Neurons in each Hidden Layer	**(26,26)**
Accuracy	**86.47%**

Table 9. Feed Forward Neural Network Hyperparameters Using GridSearchCV Approach, 'adam' Solver and 'Relu' Activationwith Its Corresponding Accuracy.

Grid search cross validation approach has not been implemented using 'tanh' activation function as the neurons take longer to fire due to the complex activation function.

4.4 RandomizedSearchCV Results

Grid search approach was found to be inefficient. It did not produce better accuracy using 'adam' than it did with 'lbfgs' without searching in any grid of hyperparameters. Instead,

randomized search cross validation is a better approach when the grid of hyperparameters searched is large, which is this case. Also, because randomized search does not evaluate every possibility, the method could be performed for every solver type. Surprisingly, using this approach was suitable and it found optimal results very fast. Due to computation efficiency, the hidden layer setup was performed in a way that the algorithm could select hidden layers with and without the same number of neurons in each layer. Below are the most significant hyperparameters using randomized search cross validation.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**adam**
L2 regularization parameter	**0.1**
Activation function	**ReLU**
Neurons in each Hidden Layer	**(17,44)**
Accuracy	**88.70%**

Table 10. Feed Forward Neural Network Hyperparameters Using RandomizedSearchCV Approach, 'adam' Solver and 'Relu' Activation with Its Corresponding Accuracy.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**sgd**
L2 regularization parameter	$\mathbf{10^{-9}}$
Activation function	**ReLU**
Neurons in each Hidden Layer	**(5,20)**
Accuracy	**86.21%**

Table 11. Feed Forward Neural Network Hyperparameters Using RandomizedSearchCV Approach, 'sgd' Solver and 'Relu' Activation with Its Corresponding Accuracy.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**lbfgs**
L2 regularization parameter	**0.1**
Activation function	**ReLU**
Neurons in each Hidden Layer	**(36,24)**
Accuracy	**91.48%**

Table 12. Feed Forward Neural Network Hyperparameters Using RandomizedSearchCV Approach, 'lbfgs' Solver and 'Relu' Activation with Its Corresponding Accuracy.

Although 'relu' activation function provides the multilayer perceptron regressor model with good accuracy, it has also been implemented using 'tanh' activation function which generally performs better and found the following results.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**Adam**
L2 regularization parameter	$\mathbf{10^{-5}}$
Activation function	**Tanh**
Neurons in each Hidden Layer	**(9,64)**
Accuracy	**86.37%**

Table 13. Feed Forward Neural Network Hyperparameters Using RandomizedSearchCV Approach, 'Adam' Solver with Its Corresponding Accuracy. *In this case, using 'tanh' as activation function.*

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**Sgd**
L2 regularization parameter	$\mathbf{10^{-4}}$
Activation function	**Tanh**
Neurons in each Hidden Layer	**(5,10)**
Accuracy	**83.08%**

Table 14. Feed Forward Neural Network Hyperparameters Using RandomizedSearchCV Approach and 'Sgd' Solver with Its Corresponding Accuracy.
In this case, using 'tanh' as activation function.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**Lbfgs**
L2 regularization parameter	**0.001**
Activation function	**Tanh**
Neurons in each Hidden Layer	**(20,1)**
Accuracy	**85.17%**

Table 15. Feed Forward Neural Network Hyperparameters Using RandomizedSearchCV Approach and 'lbfgs' Solver with Its Corresponding Accuracy.
In this case, using 'tanh' as activation function.

It can be inferred from the presented tables above that randomized search cross validation approach does not make a lot of difference regarding accuracy using ‘tanh’ instead of ‘relu’. The accuracy of ‘adam’ solver decreases as well as ‘sgd’. Also, due to the more complexity of ‘tanh’ activation function the optimization approach takes longer than using the linear activation function, ‘relu’. The performance regarding the ‘lbfgs’ solver does not increase either.

Chapter 5. **Best Configurations Analysis**

This chapter compares the accuracy obtained with respect to the solver type used (adam, sgd or lbfgs) in the model using statistical measurements and regression graphs. In the following sections, the best structure of each solver using the modeled artificial feed forward neural network is examined.

5.1 Adam Solver

The best structure of the artificial neural network using 'adam' solver results in a regularization parameter value of 10^{-9}, 4 neurons in each of the two hidden layers and 'relu' activation function. This structure of the artificial neural network has been obtained using the first method, trial and error with different hyperparameters. This configuration provided an accuracy of 90.5% and the following error histogram compares the predicted neutron multiplication factor by the neural network and the one output by VBUDS3.

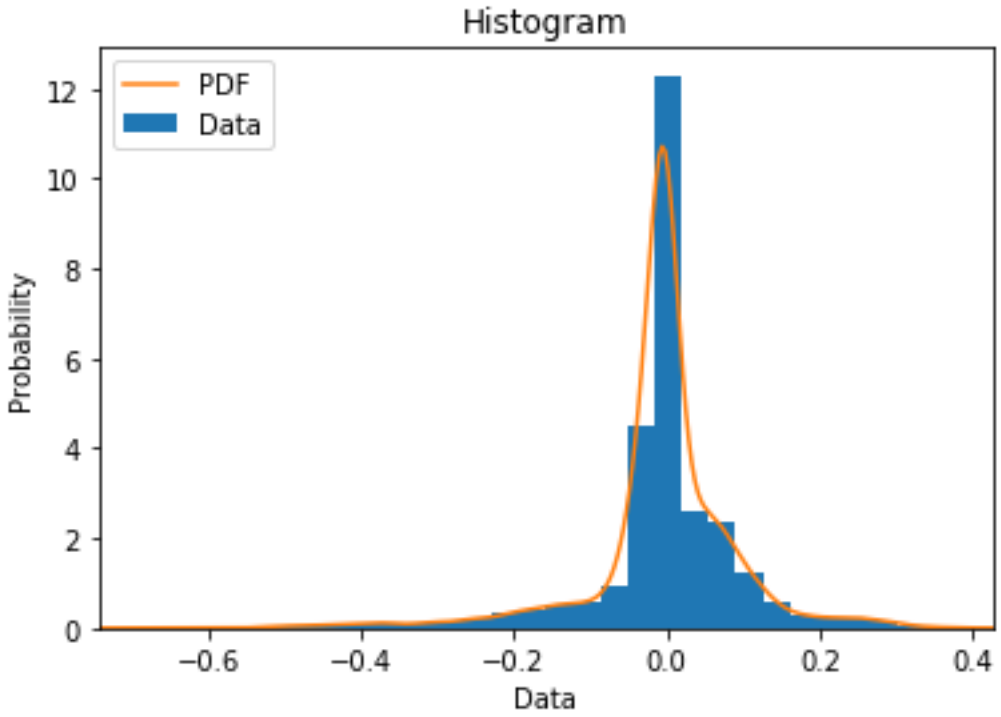

Figure 25. Error Histogram Using 'adam' Solver with 10^{-9} As Regularization Term and 4 Neurons in Each Hidden Layer. *The set up activation function was 'relu'.*

It can be noticed that the error histogram assimilates a normal distribution and it is asymmetrical to its right side. Most of the density of the histogram is close to zero, which means that the deviation of the predicted neutron multiplication factors is not high. A scatter plot is useful to know which of the neutron multiplication values are predicted correctly and which ones are deviated. The following scatter plot shows the predicted values with respect to the real values.

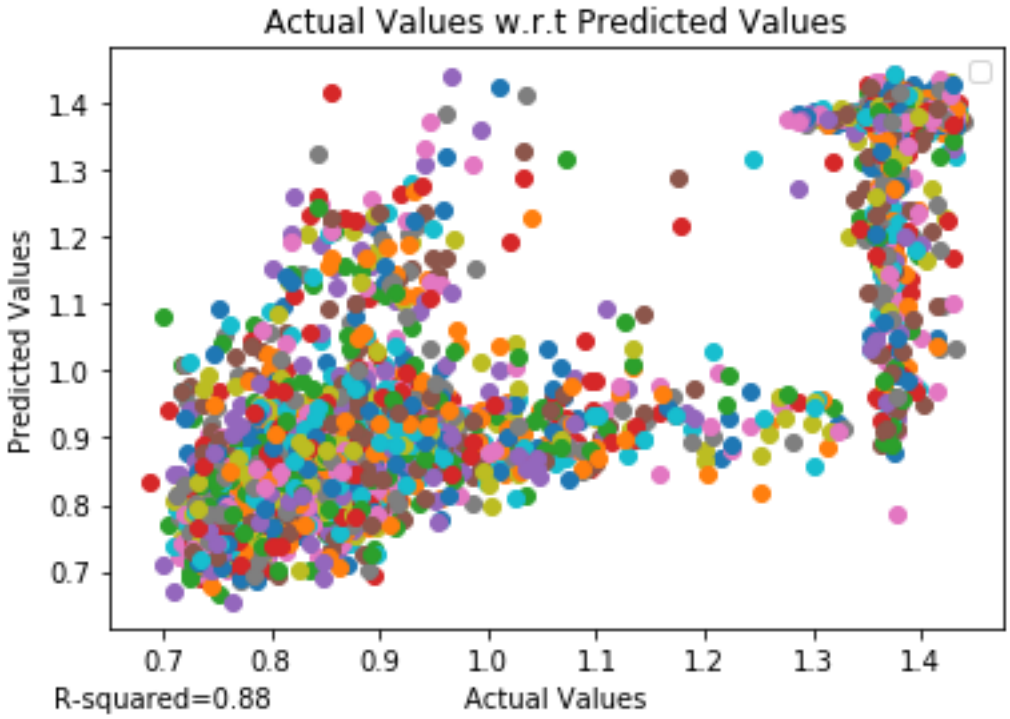

Figure 26. Scatter Plot of the Actual Values with Respect to the Predicted Neutron Multiplication Factor Values Using 'adam' Solver with 10^{-9} ***As Regularization Term and 4 Neurons in Each Hidden Layer.*** *There is a positive relationship between actual and predicted values.*

The scatter plot shows a positive relationship between the actual values and the predicted values using 'relu' activation function. There are some cases where there is more density of dots than others. The areas of the plot where the prediction is accurate are the bottom left and the top right. There is more data density than in central regions of the plot. The determination coefficient (R-squared) is a relevant measure in the study because it is a statistical measure which represents the proportion of the variance for a dependent variable that is

explained by an independent variable [26]. Using 'adam' solver the determination coefficient is 88% which means that the predictions are well fitted to the real values. Also, knowing the loss value with respect to the number of cycles that the network takes using the training data set is relevant in the study. The loss function using 'adam' solver is represented below.

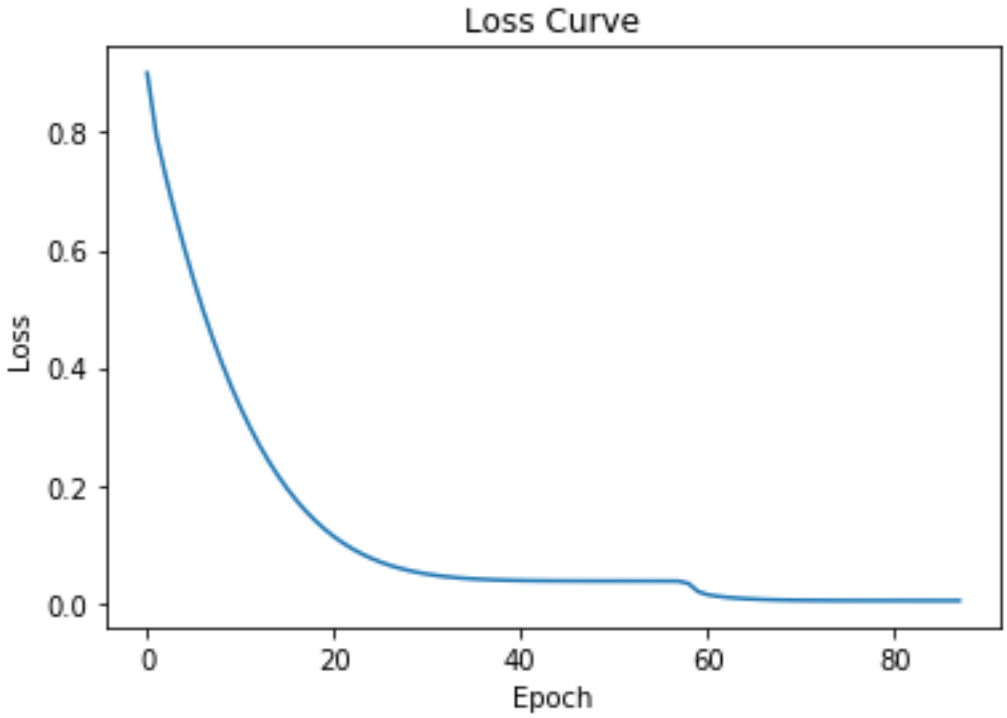

Figure 27. Computed Loss with Respect to the Number of Epochs Performed by the Neural Network Using 'adam' Solver with 10^{-9} As Regularization Term and 4 Neurons in Each Hidden Layer.

[26] Hayes, "R-Squared."

The feed forward artificial neural network is able to learn the parameters after 60 cycles using the training data. Once the neural network runs for 20 cycles, the loss value is low, but it is not until reaching 60 where this loss does not decrease more. Learning curves are widely used in machine learning and it allows the user to discover the behavior of the network. The learning curve using training score and cross validation score is shown below.

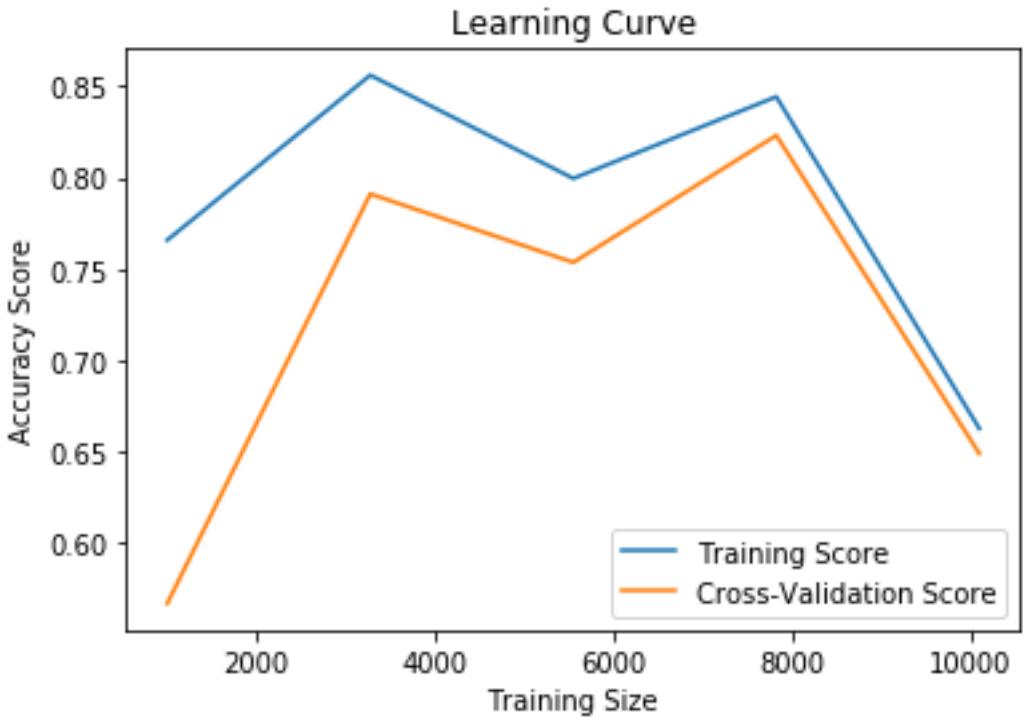

Figure 28. Learning Curve Comparing Training Score and Cross-Validation Score Using 'adam' Solver with 10^{-9} As Regularization Term and 4 Neurons in Each Hidden Layer. *There is a decrease in the score which needs to be tackled by reducing the training size to 8,000 samples.*

Learning curves show the validation and training score of an estimator depending on the number of training samples used. In this case, it can be observed in the graph that the training and validation data converge to the same score using 8,000 training samples. When the model starts training more data, the score becomes to decrease which means that reducing the number of training samples could improve the accuracy of the artificial neural network.

5.2 Sgd Solver

Using stochastic gradient descent approach, the best performance of the artificial neural network was 86.67% using a regularization term of 10^{-9} and 1 neuron in each hidden layer. Also, the neurons were activated by 'relu' functions. This configuration of the neural network is found the same way as the best configuration using 'adam' solver, initializing the regularization term manually. The error histogram of the performance of the network is shown in the figure below.

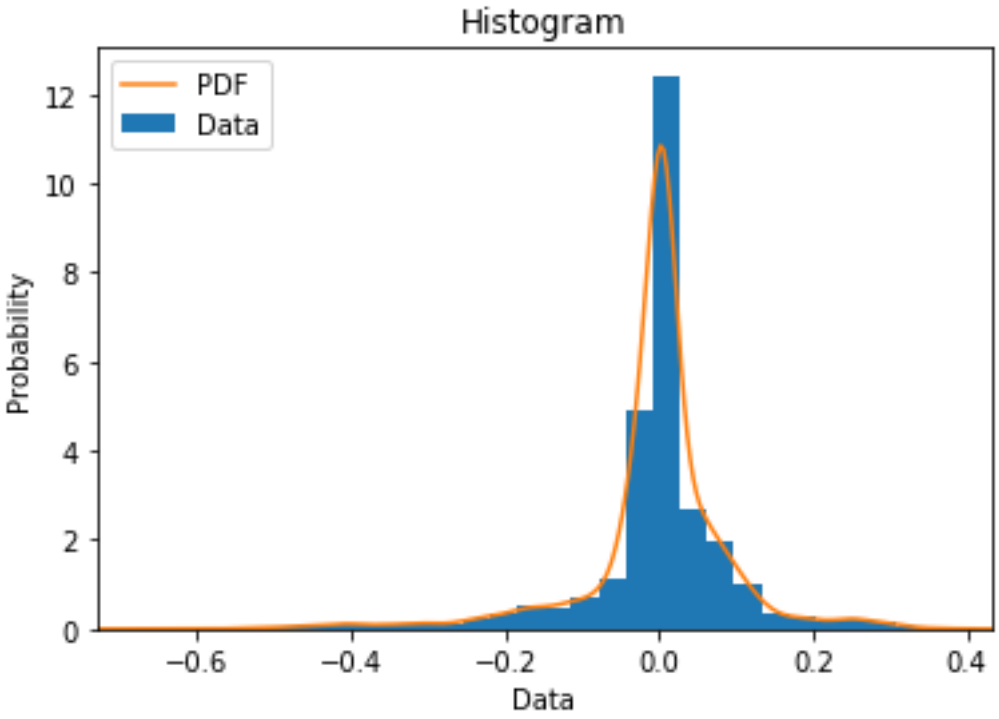

Figure 29. Error Histogram Using 'sgd' Solver with 10^{-9} As Regularization Term and 1 Neuron in Each Hidden Layer.

As it can be seen in the histogram, it is very similar to the histogram used with 'adam' solver. This means that the predicted error of the artificial neural network is concentrated around the zero value and the resulting accuracy of both models is very similar. The histogram is symmetric to its right side. Also, the scatter plot and the determination coefficient provide this project with useful information. The resulting scatter plot using this multilayer perceptron regressor model appears as follows.

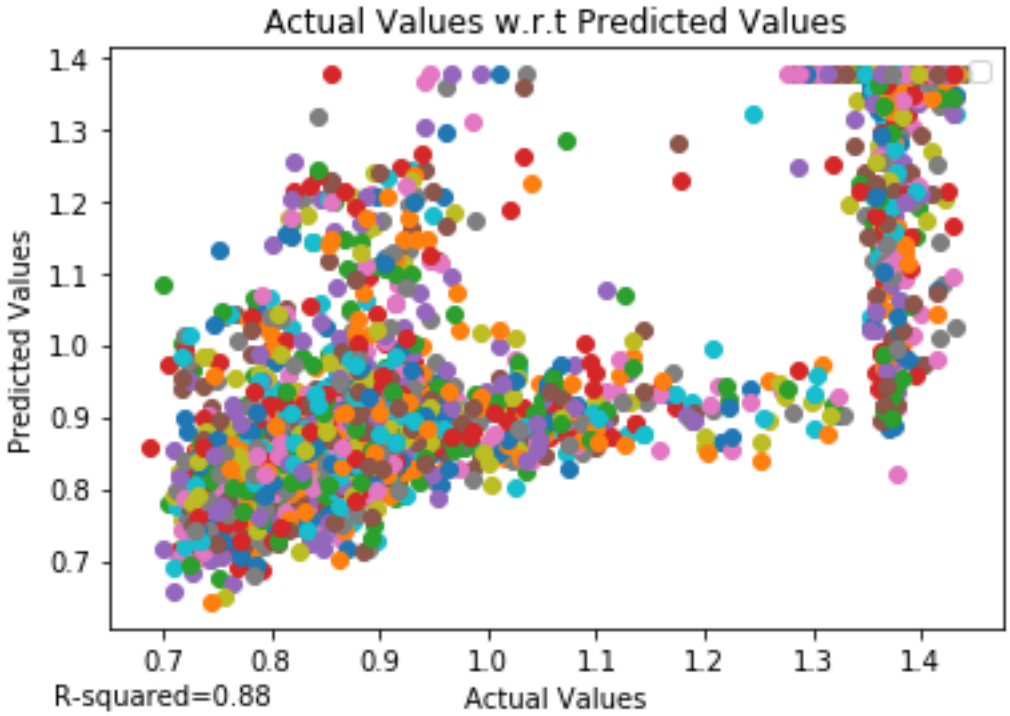

Figure 30. Scatter Plot of the Actual Values with Respect to the Predicted Neutron Multiplication Factor Values Using 'sgd' Solver with 10^{-9} As Regularization Term and 1 Neuron in Each Hidden Layer. *There is a positive relationship between actual and predicted values.*

The scatter plot using 'sgd' solver assimilates to the previous one, where the areas having more density of data are the top right and the bottom left. The neutron multiplication values in the bottom left side of the plot are predicted fairly well, as the model follows a

straight line having the actual values and the predicted values positive relationship. Once again, the model has a determination coefficient of 88% and the data does not have much noise. The loss function using stochastic gradient descent model is provided below.

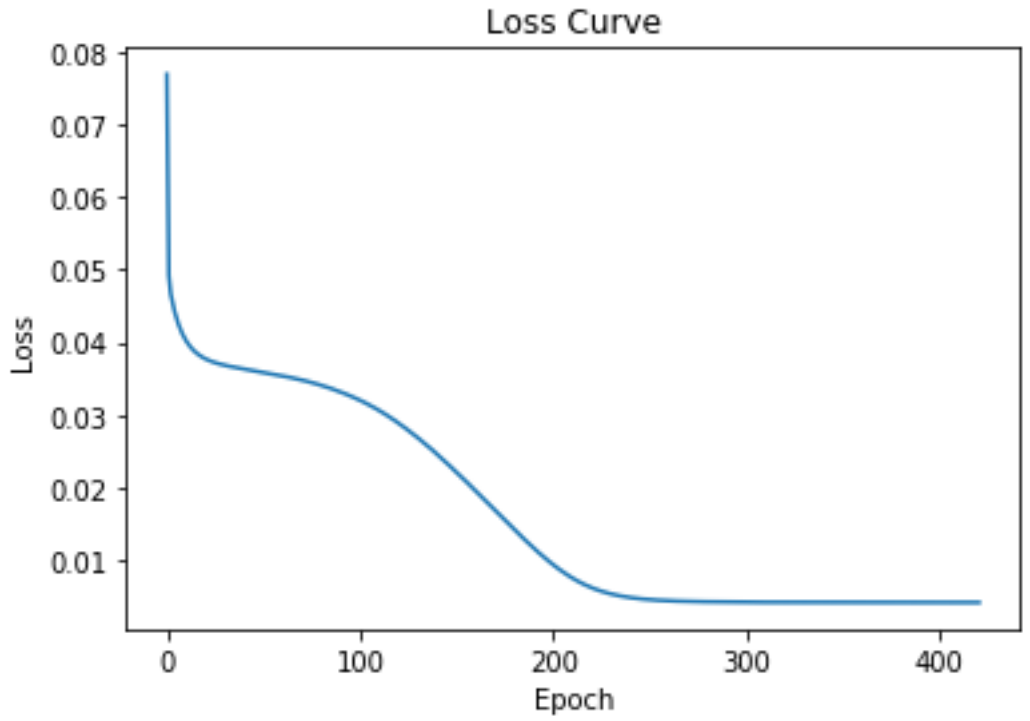

Figure 31. Computed Loss with Respect to the Number of Epochs Performed by the Neural Network Using 'sgd' Solver with 10^{-9} As Regularization Term and 1 Neuron in Each Hidden Layer.

It is interesting to notice the initial value of the loss using this model. In this case, the initial loss value is 0.08 whereas in the previous case, it started with 0.8. However, using

stochastic gradient descent, the loss values decreases very slowly which means that the neural network needs to perform above 200 cycles using the training data to decrease the loss significantly. This tells us why stochastic gradient descent solver is slower than 'adam'. "The multilayer perceptron model used with Scikit-Learn trains the data using Stochastic Gradient Descent, Adam or L-BFGS". "The stochastic gradient descent optimizer updates the parameters using the gradient of the loss function with respect to a parameter that needs adaptation" [27].

$$w \leftarrow w - \eta \cdot (\alpha \cdot \frac{\partial R(w)}{\partial w} + \frac{\partial Loss}{\partial w})$$

"Where η stands for the learning rate which controls the step decrease in the minimization function and w the corresponding input weight to the neuron".

The learning history of the solver using training score and cross validation score appears as follows.

[27] "1.17. Neural Network Models (Supervised) — Scikit-Learn 0.23.0 Documentation," 5.

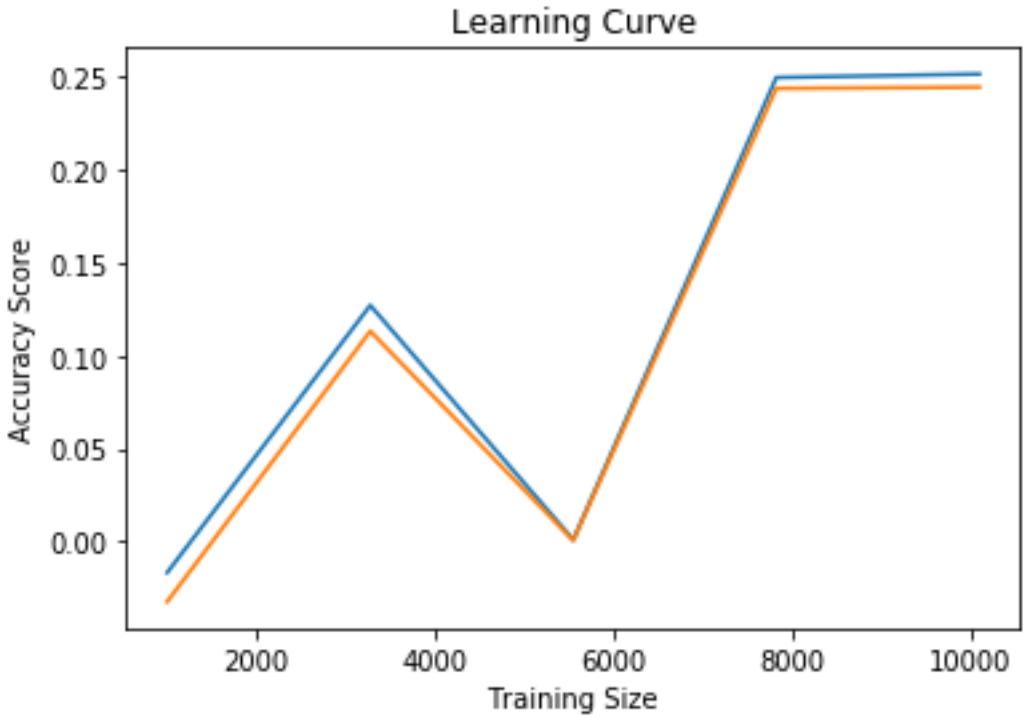

Figure 32. Learning Curve Comparing Training Score and Cross-Validation Score Using 'sgd' Solver with 10^{-9} As Regularization Term and 1 Neuron in Each Hidden Layer. *There is no improvement in the accuracy score after 8,000 training samples are used.*

The accuracy score of the training set (blue line) and the accuracy score of the cross-validation set (red line) appear to be close together along the training samples. The accuracy score of this model does not improve when the training set reaches 7,500 data points. This means that adding more data to this multilayer perceptron regressor model will not improve the accuracy of the model, instead it will remain the same. There is underfitting in the left side of the graph where the training score and cross validation score

are almost the same, but this problem is solved when both scores reach the same value using all the training data.

5.3 Lbfgs Solver

The structure of the feed forward neural network performed best using ‘lbfgs’ solver and ‘tanh’ activation instead of ‘relu’ giving a total of 94.90% infinite neutron multiplication factors predicted correctly. The hyperparameters which resulted in the best configuration of the network are regularization term 10^{-9} and 13 neurons in each hidden layer. The error of this model is not perfect but most data density is around zero as the histogram below shows.

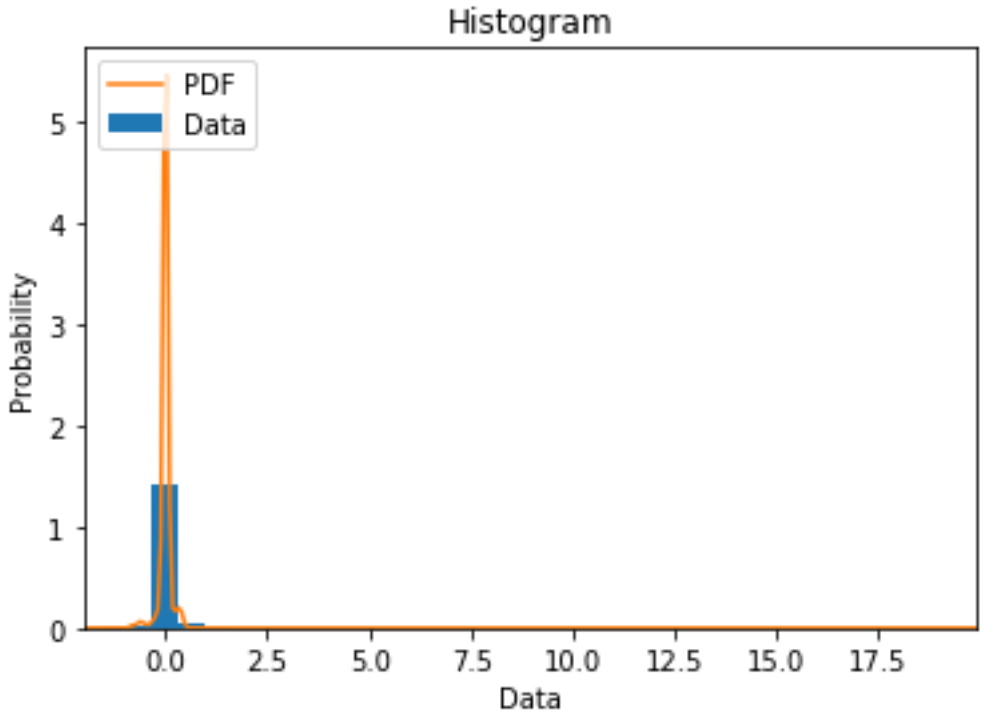

Figure 33. Error Histogram Using 'lbfgs' Solver with 10^{-9} As Regularization Term and 13 Neurons in Each Hidden Layer. *The model used 'tanh' as activation function.*

In this multilayer perceptron regressor model, we could approximate the histogram to be symmetric. It is unimodal where the most repeated error value is zero. The outliers are represented better in the scatter plot. The scatter plot of this feed forward neural network structure is below.

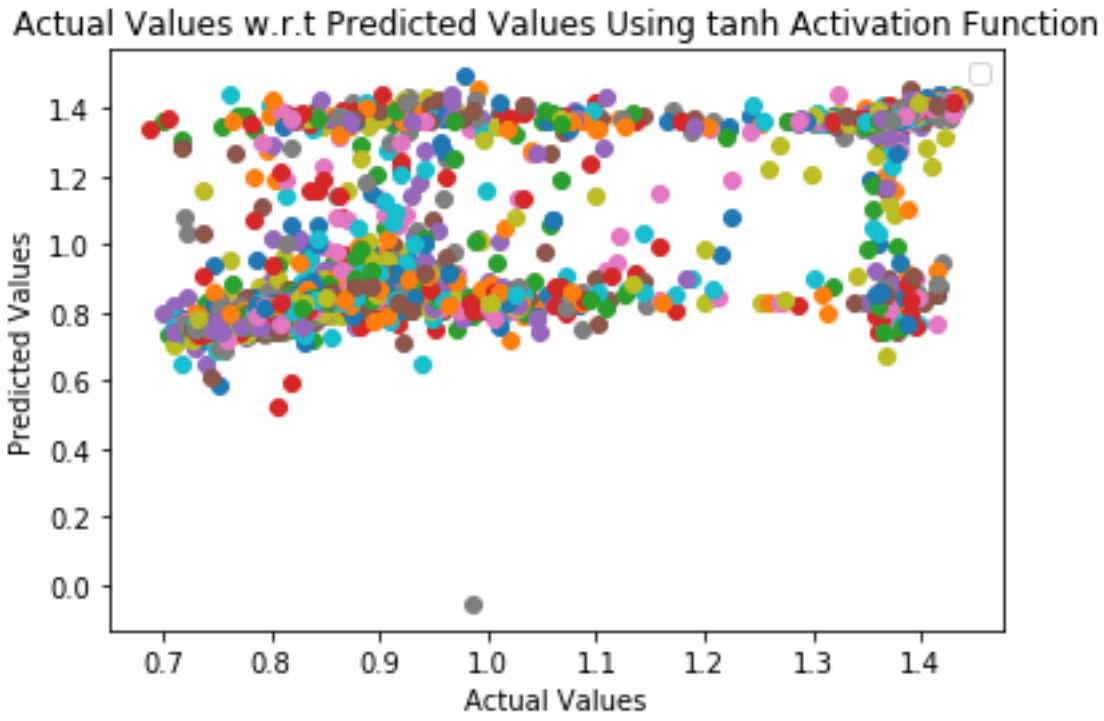

Figure 34. Scatter Plot of the Actual Values with Respect to the Predicted Neutron Multiplication Factor Values Using 'lbfgs' Solver with $\mathbf{10^{-9}}$ ***As Regularization Term and 13 Neurons in Each Hidden Layer.*** *There is a positive relationship between actual and predicted values.*

Although the accuracy of this model provides a better accuracy, the determination coefficient appears to be lower than the previous cases, 78%. The model is found to be more accurate and it can be observed in the scatter plot, as there are more predicted values correctly. However, there are some outliers in the model. The most drastic ones are located in the upper left corner where the actual neutron multiplication factors are low

and the predicted are high and on the center low where the actual value is medium range and the predicted values is low.

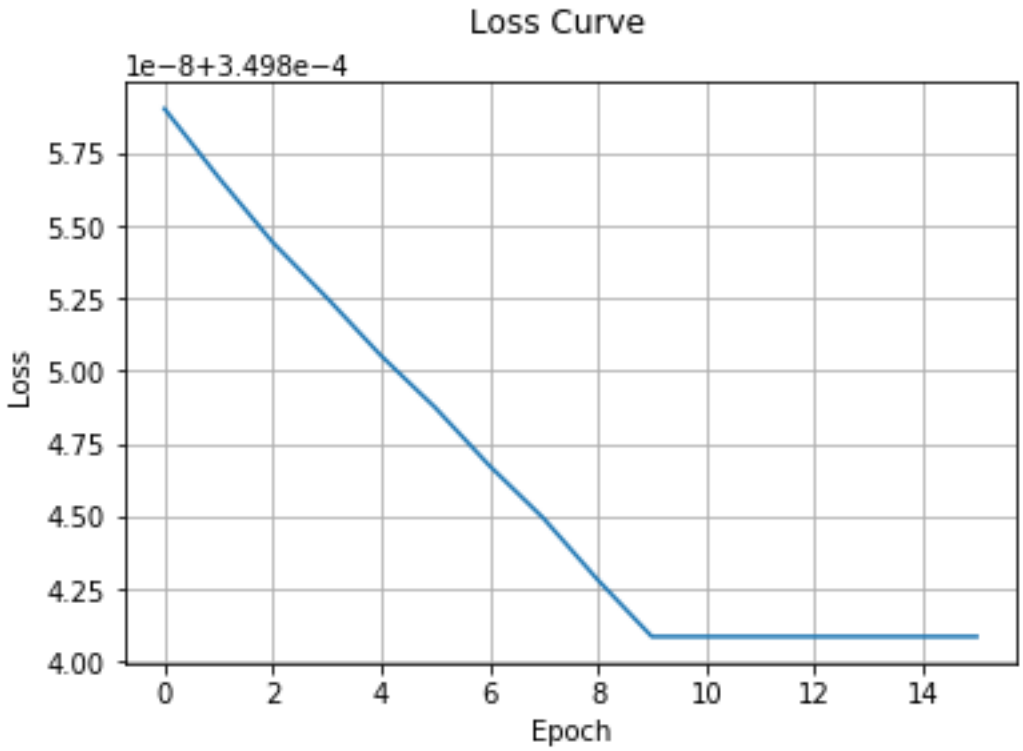

Figure 35. Computed Loss with Respect to the Number of Epochs Performed by the Neural Network Using 'lbfgs' Solver with $\mathbf{10^{-9}}$ ***As Regularization Term and 13 Neuron in Each Hidden Layer.***

'Lbfgs' solver does not have the loss curve function implemented in Scikit Learn. In order to plot the decay of the model with respect to each epoch, it has been computed the mean squared error of each iteration and plotted the results. It can be observed in the graph that the loss already starts with a low value, $3.49859 \cdot 10^{-4}$ and only decreases slightly until

reaching epoch 9. The optimizer in the family of quasi-Newton methods uses the mean squared error to compute the loss in each iteration that the neural network performs.

$$MSE = \frac{1}{n} \cdot \sum (y - \hat{y})^2$$

Where n represents the number of observations, y the infinite neutron multiplication factor for a given energy group structure using the collision probability code and $\hat{y}$ is the estimated value of the neutron multiplication factor computed by the artificial neural network.

Although this configuration provides the best accuracy, the learning curve states that the model could be improved.

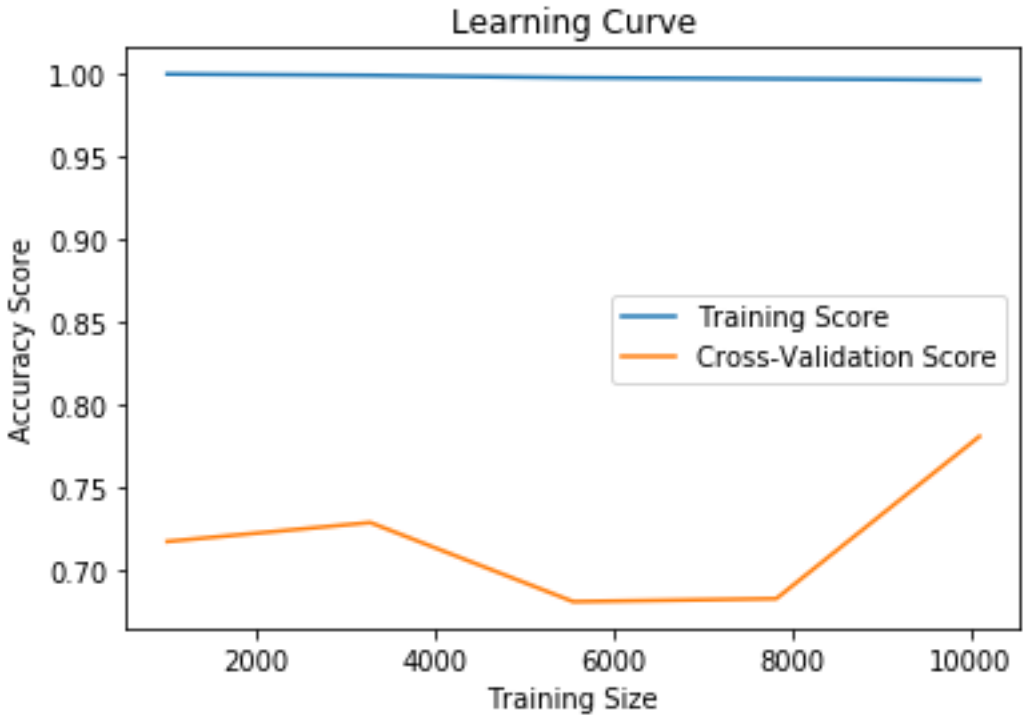

Figure 36. Learning Curve Comparing Training Score and Cross-Validation Score Using 'lbfgs' Solver with 10^{-9} ***As Regularization Term and 13 Neurons in Each Hidden Layer.*** *There is a gap between the training score and the cross-validation score, which states that this model suffers overfitting.*

It can be observed in the learning curve that the cross-validation score does not reach the same level as the training score, there is a large gap between the two. Particularly, this structure is suffering from overfitting and there are some parameters which could be modified in order to obtain better accuracy and eliminate the overfitting such as the regularization term or the complexity of the neural network." The model is learning the training data too well and performs poorly when using the test set". "Overfitting can

produce misleading R-squared values, regression coefficients and p-values" [28]. Also, the training size could be increased over 70% to increase the model's score. It has been found that increasing the regularization term over zero helps the model decrease overfitting. Also, reducing the fraction of data used helps to increase the overall accuracy. The most balanced parameters which provided reasonable accuracy reducing overfitting are shown in the table below. The selected architectures are modeled using 'tanh' activation function, which provides best accuracy using 'lbfgs' solver.

Machine Learning Algorithm Hyperparameters

Artificial Neural Network Parameters	**Value**
Solver Type	**lbfgs**
L2 regularization parameter	**6**
Activation function	**Tanh**
Neurons in each Hidden Layer	**(55,29)**
Accuracy	**88.09%**

Table 16. Balanced Configuration of the Artificial Neural Network Regarding Overfitting and Accuracy.

[28] "Overfitting Regression Models."

Although the simulated annealing optimization strategy did not help much in previous cases, I have used this approach to find the best model with a good compromise between accuracy and overfitting. The resulting model hyperparameters are shown in the table above and it can be observed in the learning curve below that the training and testing scores reach almost the same value, leading to a reduction in overfitting.

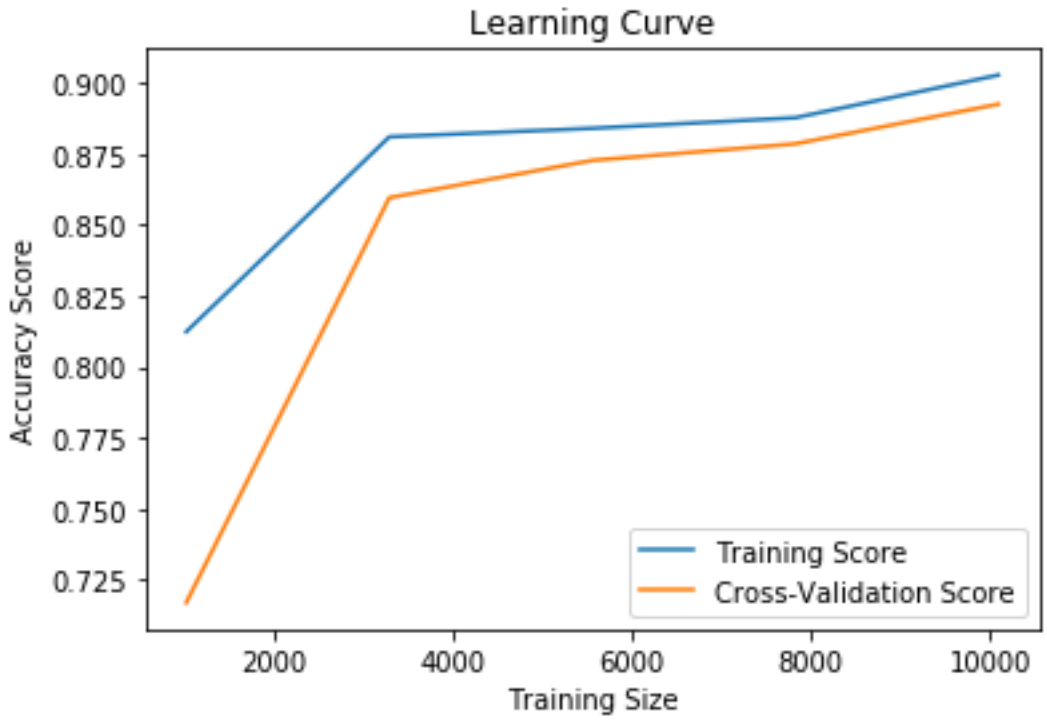

Figure 37. Training and Cross Validation Scores with Respect to Training Size Using Optimal Hyperparameters Found with Simulated Annealing in Order to Reduce Overfitting.

As mentioned in previous sections, it has been found for this multilayer perceptron regressor that decreasing de regularization term improves accuracy of the neural network. But using the learning curve I have realized that, although the model provides better accuracy, there is overfitting. Increasing the regularization term over zero improves significantly signs of overfitting, leading to a good compromise between accuracy of the feed forward neural network and overfitting shown in the previous table. If we use this structure of the artificial neural network as the most efficient, the weights and biases are shown below.

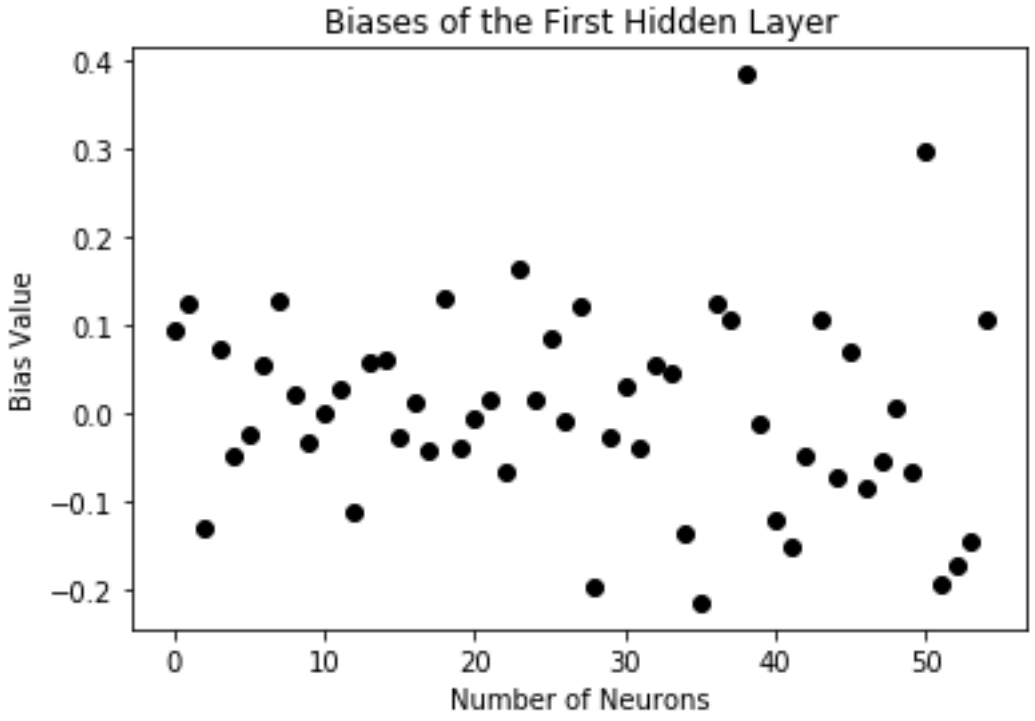

Figure 38. Bias Value of Each Neuron in the First Hidden Layer.

The above distribution shows the bias values of the first hidden layer, which consists of 55 neurons. There is one value of bias associated with each node performing the study considering the most efficient configuration of the neural network using simulated annealing.

Figure 39. Bias Value of Each Neuron in the Second Hidden Layer.

In this case, the optimal multilayer perceptron regressor model suggests that the second hidden layer of the network consists of 29 neurons which biases are plotted in the above diagram. The output layer consists of one neuron which provides a bias value of 0.405. Due to the random weight initialization the accuracy and the biases of each neuron of the neural network varies slightly." The input weights to each neuron of the network is represented with a list in Scikit-Learn where the ith element in the list represents the weight matrix corresponding to layer i "[29]. In each matrix, the columns represent the number of neurons in the corresponding hidden layer and the rows the number of neurons in the previous hidden layer. Each neuron of a certain layer receives the output weights of each of the neurons in the previous layer. Meaning that in this architecture of the artificial neural network the matrices have the following dimensions.

Artificial Neural Network Weights' Matrices	Dimension
Hidden Layer 1	**(199,55)**
Hidden Layer 2	**(55,29)**
Output Layer	**(29,1)**

Table 17. Matrices' Weights Dimensions of the Optimized Neural Network.

[29] "Sklearn.Neural_network.MLPRegressor — Scikit-Learn 0.23.0 Documentation."

The following diagram represents the weights' values of the output layer.

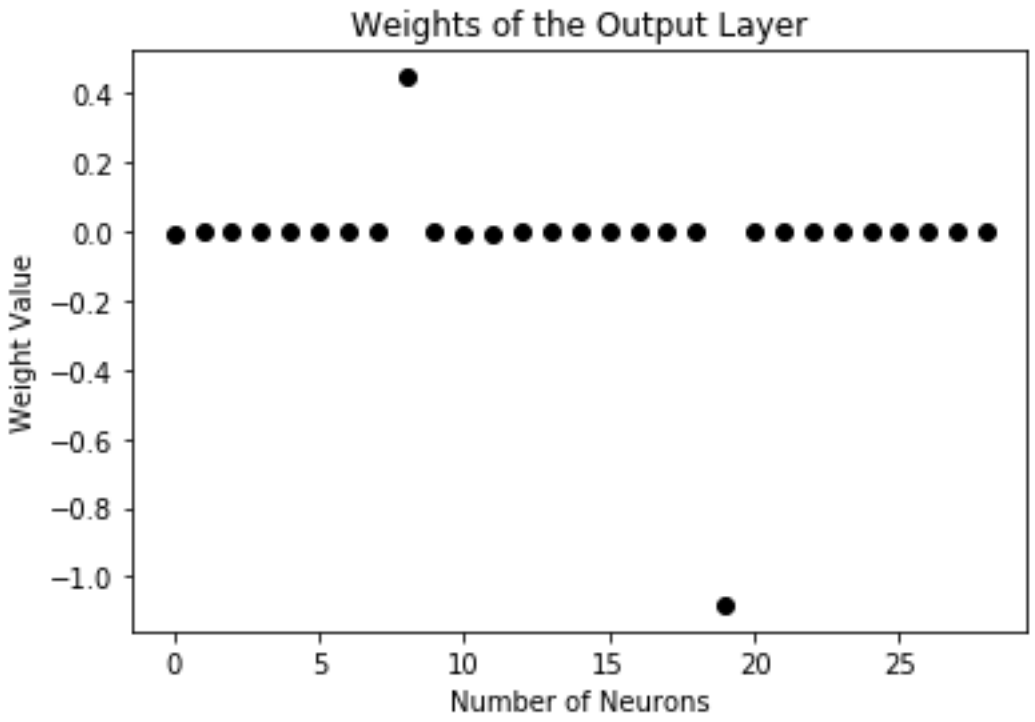

Figure 40. Weight Values Considering the Output Layer.

The output layer of the artificial neural network consists of one neuron and its value depends on the bias of that neuron and the incoming weights from the previous hidden layer. This layer is made of 29 neurons and it can be observed in the scatter plot that the significant incoming weights belong to neuron 8 (firing a value of 0.448) and neuron 19 (firing a value of -1.081).

Chapter 6. **Integrating the Sustainable Development Goals**

This chapter will be focused on integrating the most relevant sustainable development goals in the project. Section 5.1 will focus on the contextualization of the main sustainable development goal applied to this case and section 5.2 will focus on secondary sustainable development goals identified and how the project interacts with the main and secondary goals.

6.1 Main Goal and Contextualization

Nuclear energy is one of the main sources of energy which is widely used. “The environmental impact of this source of energy can be less damaging than other sources such as coal energy”. However, as it is known, nuclear energy has damaging effects too. “First, the wrong use of nuclear energy and nuclear reactors can lead to nuclear catastrophes such as the well-known Chernobyl disaster”. “Once the nuclear reactor is operating, the emitted amounts of carbon dioxide are low, but the most relevant environmental impact of nuclear reactors is regarding how the fuel is extracted”. “The process of mining uranium can release high amounts of carbon dioxide” [30]. In this

[30] “How Does Nuclear Energy Affect the Environment?”

project, the main sustainable development goal identified is goal 13: “Climate Action”. It is vital that the whole world takes urgent action to face climate change and the environmental impact of human processes. One way in which environmental damage could be reduced is by running nuclear reactors in a more efficient and clean way. This means using different kinds of fuel or coolant to operate nuclear reactors. The first action which needs to be considered is the “improvement in education, awareness-raising and human and institutional capacity on climate mitigation, adaptation, impact reduction and early warning” [31].

6.2 Secondary Sustainable Development Goals and Interaction with the Project

“The radioactive waste is one of the main concerns regarding environmental impact of nuclear reactors because this waste can last for thousands of years having critical effect in the surrounding air, which can be lethal to human beings”.” There are many ways in which the nuclear waste can be managed, one of them include the storage of radioactive waste in the plant”. However, due to storage limitations this is not very efficient when running a nuclear reactor for a long time.” The nuclear waste needs to be relocated and, but the process can be costly and not fully eliminate the nuclear waste”. “It is said that

[31] “#Envision2030 Goal 13: Climate Action | United Nations Enable.”

there are currently around 250,000 tons of nuclear waste distributed across 14 countries" [32]. There is another issue regarding the cooling system of nuclear reactors, which is used to prevent the core from overheating. Cooling water is sometimes provided by the ocean which cools down the core and then returns back. "This leads to an increase in dead fish and plants". The nuclear waste generation and the water pollution constitute secondary concerns regarding the sustainable development goals. In particular, goal 12: "Ensure sustainable consumption and production patterns". It is essential to manage the renewable resources sustainably reducing through prevention, reduction, recycling, and reuse.

In order to combat the radioactive waste and pollution, this project simulates a pressurized light water reactor. The simulation of these types of reactors consider two separate loops. One of the constitutes the cooling system and the other forms the nuclear core. Performing the simulations in light water reactors allows the cooling water to never make contact with nuclear fuel, avoiding any water radioactive contamination. Simulation is key regarding environmental impact. In the case of this thesis, the prediction of the infinite neutron multiplication factor can determine the state in which the modeled nuclear reactor is operating. If the modeled k_∞ is equal to one, the nuclear reactor would be operating in a critical way meaning that there is no change in the neutron population. However, if the parameter is greater than 1 after simulation, the reactor is operating in a supercritical way and the neutron population would be increasing. Neutron population increasing inside the core can be beneficial in regard to the number of interactions. The more neutron population, the more probability of

[32] "Storage of Nuclear Waste a 'Global Crisis': Report."

interaction between neutrons and matter. But this can be damaging as well. If the neutron population increases, there is more chance of nuclear matter being wasted. The prediction of the neutron multiplication factor in artificial neural networks is, not only faster to simulate than in a complex collision probability model, but it could also be useful for simulating other relevant parameters of the nuclear reactor which affects the environment.

Chapter 7. Conclusion and Future Scope

In this chapter there is a final discussion and conclusion of the project in section 7.1 and how it could the model be improved in future approaches in section 7.2

7.1 Final Discussion

Machine learning algorithms can be a very useful tool when solving problems of large magnitudes. It has been observed in the project that a feed forward artificial neural network helped us predict the infinite neutron multiplication factor given a random 20-group energy structure. As a conclusion, it has been shown that artificial neural networks can be used for reactor analysis. In particular, this project showed how a feed forward neural network can predict a parameter of a nuclear reactor, the neutron multiplication factor. Although neural networks need many data, future and more complex reactor analysis could be performed using machine learning. This thesis used three hyperparameter optimization techniques to obtain the best artificial neural network structure, but it has been found that the best accuracy happened selecting the hyperparameters using trial and error. However, the weights of the input neurons of the network are set randomly by default, which means that a certain model will have an accuracy varying in a range of values. This allowed the optimization techniques implemented to be useful in some cases. The feed forward neural network provided with an alternative structure providing better accuracy. The most successful optimizer was

random search cross validation approach, having a lower time of finding a better structure than grid search cross validation or simulated annealing. Simulated annealing approach has also been found to be useful providing optimized models without overfitting. Overall, the expectations of the neural network to learn relevant parameters of the problem was high, as it demonstrated giving the best accuracy of nearly 95%.

7.2 Future Work

It has been demonstrated that relevant parameters of a nuclear reactor can be predicted using machine learning and feed forward artificial neural networks. This project shows the potential of machine learning as well. The infinite neutron multiplication factor can be calculated using the collision probability code because the nuclear reactor model is not very complicated and used only four nuclides. However, simulating a much more complicated nuclear reactor can be tough and running the collision probability code could be inefficient. Instead, the use of machine learning and neural networks allows to model more complicated nuclear reactors. The feed forward artificial neural network could be used in future work for predicting other relevant parameters of nuclear reactors.

Chapter 8. **Appendix**

8.1 Used Symbols

k_∞	Infinite Neutron Multiplication Factor
Φ	Neutron Flux
n	Neutron Density
v	Neutron Velocity
w_j	Weight of Input j
b	Neuron's Bias
S^r	Training Sample Input
E^r	Desired Output of the Training Sample
E	Energy of Incoming Neutron

Φ_g^i Neutron Flux in Each Energy Group g and Region i

V^i Volume of Region i

$\sum_{t,g}^{i} \Phi_g^i$ Total Cross Sections

$\sum_{f,g'}^{j} \Phi_{g'}^{j}$ Fission Cross Sections

$\sum_{s,g'g}^{j} \Phi_{g'}^{j}$ Scattering Kernel from Group g' to Group g

χ_g Fission Spectrum

p_g^{ji} Probability that a Neutron has to Appear in Group g and Region j After a Collision or Reaction Will Have Its Next Collision in Region i

L Matrix of Removal Minus Scattering Reactions

F Matrix of Fission Reactions

N_i Atomic Number Density of Nuclide i

$\sigma_{f,j}$	Fission Cross Section of Nuclide j
$\sigma_{c,j}$	Neutron Capture Cross Section of Nuclide j
a_{ji}	Yield of Nuclide i Produced by Fission in Nuclide j
b_{ji}	Probability that Reactions in Nuclide j Produce Nuclide i
$\sigma_{a,i}$	Neutrons Absorbed Reactions in Nuclide i
λ_j	Nuclide j's Decay Constant
c_{ji}	Branching Ratio to Nuclide i
A	Neutron Reaction Matrix
B	Decay Matrix
N	Vector of Atomic Number Densities
y	Infinite Neutron Multiplication Factor Given a Random Energy Group Structure Using the Collision Probability Code

$\hat{y}$ Infinite Neutron Multiplication Factor Predicted by the Neural Network Given a Random Energy Group Structure

8.2 List of Tables

8.3 List of Figures

COLORADO SCHOOL OF MINES

Mechanical Engineering

Appendix

8.4 References

"1.17. Neural Network Models (Supervised) — Scikit-Learn 0.23.0 Documentation." Accessed May 14, 2020. https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

Abdulla, Ahmed. "The Demise of US Nuclear Power in 4 Charts." The Conversation. Accessed May 8, 2020. http://theconversation.com/the-demise-of-us-nuclear-power-in-4-charts-98817.

"Artificial Neural Network." In *Wikipedia*, May 10, 2020. https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=955889305.

Berry, Jessica J, and Andrew G Osborne. "A COLLISION PROBABILITY AND DEPLETION MODEL FOR RAPID SCOPING AND OPTIMIZATION OF NUCLEAR REACTORS," n.d., 8.

Brownlee, Jason. "A Gentle Introduction to the Rectified Linear Unit (ReLU)." *Machine Learning Mastery* (blog), January 8, 2019.

https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

Chiesa, Davide, Ezio Previtali, and Monica Sisti. "Bayesian Statistics Applied to Neutron Activation Data for Reactor Flux Spectrum Analysis." *Annals of Nuclear Energy* 70 (August 1, 2014): 157–68. https://doi.org/10.1016/j.anucene.2014.02.012.

Deinert, Mark, and Erich Schneider. "A Multi-Region Collision Probability Method For Determining Neutron Spectra and Reaction Rates," n.d., 142.

"#Envision2030 Goal 13: Climate Action | United Nations Enable." Accessed May 12, 2020. https://www.un.org/development/desa/disabilities/envision2030-goal13.html.

Hayes, Adam. "R-Squared." Investopedia. Accessed May 25, 2020. https://www.investopedia.com/terms/r/r-squared.asp.

Sciencing. "How Does Nuclear Energy Affect the Environment?" Accessed May 12, 2020. https://sciencing.com/nuclear-energy-affect-environment-4566966.html.

Jain, Pawan. "Complete Guide of Activation Functions." Medium, May 8, 2020. https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044.

Liu, Danqing. "A Practical Guide to ReLU." Medium, November 30, 2017. https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7.

LLC, Cogito Tech. "Understanding the Importance Of Training Data In Machine Learning." Medium, August 26, 2019. https://medium.com/@cogitotech/understanding-the-importance-of-training-data-in-machine-learning-da4235332904.

Massone, Mattia, Fabrizio Gabrielli, and Andrei Rineiski. "SIMMER Extension for Multigroup Energy Structure Search Using Genetic Algorithm with Different Fitness Functions." *Nuclear Engineering and Technology*, Special Issue on International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering 2017 (M&C 2017), 49, no. 6 (September 1, 2017): 1250–58. https://doi.org/10.1016/j.net.2017.07.012.

"Neutron Flux." In *Wikipedia*, April 16, 2020. https://en.wikipedia.org/w/index.php?title=Neutron_flux&oldid=951274139.

Nielsen, Michael A. "Neural Networks and Deep Learning," 2015. http://neuralnetworksanddeeplearning.com.

Osborne, Andrew G., Timothy A. Smith, and Mark R. Deinert. "Comparison of Actinide Production in Traveling Wave and Pressurized Water Reactors." *Proceedings of GLOBAL 2013: International Nuclear Fuel Cycle Conference - Nuclear Energy at a Crossroads*, n.d.

Statistics By Jim. "Overfitting Regression Models: Problems, Detection, and Avoidance," May 26, 2017. http://statisticsbyjim.com/regression/overfitting-regression-models/.

"Simulated Annealing." In *Wikipedia*, May 22, 2020. https://en.wikipedia.org/w/index.php?title=Simulated_annealing&oldid=958116450.

"Sklearn.Model_selection.GridSearchCV — Scikit-Learn 0.23.1 Documentation." Accessed May 23, 2020. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

"Sklearn.Neural_network.MLPRegressor — Scikit-Learn 0.23.0 Documentation." Accessed May 16, 2020. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.

"Storage of Nuclear Waste a 'Global Crisis': Report." Accessed May 12, 2020. https://phys.org/news/2019-01-storage-nuclear-global-crisis.html.

Upadhyay, Yash. "Feed Forward Neural Networks." Medium, March 8, 2019. https://towardsdatascience.com/feed-forward-neural-networks-c503faa46620.

Watterson, J. "The Watt Distribution (Spectrum)," 2007, 6.

"What Is Machine Learning (ML) and Why Is It Important? | NetApp." Accessed May 13, 2020. https://www.netapp.com/us/info/what-is-machine-learning-ml.aspx.

Worcester, Peter. "A Comparison of Grid Search and Randomized Search Using Scikit Learn." Medium, June 6, 2019. https://blog.usejournal.com/a-comparison-of-grid-search-and-randomized-search-using-scikit-learn-29823179bc85.

Yi, Ce, and Glenn Sjoden. "Energy Group Structure Determination Using Particle Swarm Optimization." *Annals of Nuclear Energy* 56 (June 1, 2013): 53–56. https://doi.org/10.1016/j.anucene.2012.12.020.

Zulkifli, Hafidz. "Understanding Learning Rates and How It Improves Performance in Deep Learning." Medium, January 27, 2018. https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10.

COLORADO SCHOOL OF MINES

Mechanical Engineering

Appendix
