UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics

Final Dissertation

Differential Equations for Feynman Integrals and Physics Informed Neural Network

Thesis supervisor
Dr. Manoj Kumar Mandal

Thesis co-supervisor
Prof. Pierpaolo Mastrolia

Candidate
Giovanni Boni

Academic Year 2023/2024

Abstract

This thesis explores a novel research line that combines Quantum Field Theory and Computer Science, by focusing on the use of Deep Learning and Neural Networks techniques for the numerical evaluation of multi-loop Feynman integrals, which are the building blocks for the calculation of Scattering Amplitudes in Perturbation Theory.
Feynman integrals are known to obey systems of first-order partial differential equations (PDEs), and owing to this property, as recently observed in the literature, their numerical evaluation can be addressed by means of Physics-Informed Neural Networks (PINNs).
Within PINNs, differential equations become an intrinsic theoretical constraint to be implemented into the machine learning process, by adding, during the training phase, a physics-motivated term to the loss function.
This approach directly accommodates the PDEs governing Feynman integrals, facilitating the learning algorithm to capture the correct solutions.
In this thesis, We begin by demonstrating the application of PINNs to solve classical differential equations, such as the hypergeometric and harmonic oscillator equations, showcasing the potential efficiencies and accuracies of PINNs.
Elaborating on the recent ideas proposed in the literature, we apply PINNs to systems of PDEs obeyed by one- and two-loop Feynman integrals, taken from scattering reactions of increasing complexity, whose analytical solutions, around four space-time dimensions, ranges from classical polylogarithms to generalised polylogarithms and to elliptic functions, and propose novel strategies to improve the NN reconstruction.
Through this exploration, we highlight the possible powerful synergy between Machine Learning and Feynman calculus, suggesting possible advancements in the area of Computational Quantum Field Theory, and more generally for problems in Applied Mathematics and Computational Science benefitting from the development of new classes of numerical solvers for PDEs.

Contents

Listing of figures

Listing of tables

Listing of acronyms

DEQ ..... Differential Equation

PDE ..... Partial Differential Equation

MAE ..... Mean Absolute Error

MSE ..... Mean Squared Error

(A)NN ..... (Artificial) Neural Network

PINN ..... Physics Informed Neural Network

AD ..... Automatic Differentiation

$\boldsymbol{N_M}$ ..... Number of Master Integrals

$\boldsymbol{N_\varepsilon}$ ..... Number of terms of the truncated Laurent expansion of a Master Integral

w.r.t. ..... with respect to

1

Introduction

1.1 State of the art and Motivation

Scattering amplitudes in Quantum Field Theories (QFT) are *complex* quantities that describe the fundamental interactions among elementary particles, and their absolute squared value are related to the quantum probability of an event to take place in Nature. Their evaluation is of paramount importance, and in the lack of a strategy to determine their value exactly, they can be computed in perturbation theory by means of Feynman diagrams. The latter are a pictorial representation of the particle scattering and correspond to mathematical function which need to be evaluated, whose complexity is related to the complexity of the graphs that increases with the number of legs (external particles), with the number of loops, and with the number of scales (particle masses and kinematic invariants, built out of the particle four-momenta). If the tree-level diagrams represent simple, rational, mathematical functions, the loop diagrams represent non trivial integrals, referred to as Feynman integrals, whose evaluation has been always representing strong motivation for advances in the understanding of fundamental physics.

The evaluation of scattering amplitudes has been recognised as one of the most profound problems in Theoretical Physics, as it has impact not only in Particle

Physics phenomenology, but also in the investigation of the properties of gauge theories, their relation to Gravity, and more recently on Gravitational Wave Physics and Cosmology. The broad range of applications is due to the ubiquity of Feynman (like) integrals that can appear also in contexts of classical as well as effective field theories.

The need of theoretical calculations more and more accurate, which can follow the advances in accuracy of experimental results, that explore novel technologies, pushed an intense progress in the so called Amplitudes research area, which from the pure Physics contexts reaches naturally out also the areas of Mathematics and Computer Science.

The main goal of this thesis is to explore a novel direction for the evaluation of multi-loop Feynman integrals, which combines modern methods from Amplitudes and computational QFT, on the one side, and Computer Sciences, on the other side.

Generally, any Feynman Diagram (and its relative Feynman Integral) admit a spinor/tensor structure, that can be decomposed in terms of basic operators with coefficients known as *form factors*. In the case of diagrams with loops, the form factors are written as combination of Scalar Integrals, therefore the evaluation of scattering amplitudes requires the calculation of those integrals, whose number, depending on the process and on the number of loops, can be tremendously large.

Multi-loop Feynman integrals are mathematically ill-defined, because they present (infrared and ultraviolet) divergences which need to be regulated. Dimensional regularization [1] is a well established regularization scheme where the space-time dimensions are analytically continued from 4 to a generic value "d", within which any divergent behaviour can be captured by singularities that appear as poles in the parameter ε $(= 4 - 2d)$, when the integrals are expanded as Laurent series around the original 4 dimensions.

Luckily, dimensionally regulated Feynman integrals are not independent, and the so-called integration-by-parts identities (IBPs) [2] can be exploited to build relations among integrals, yielding the decomposition of any integral into a finite, independent set of them, dubbed *Master Integrals* (MIs) – which can be considered as the generators of a vector space [3, 4, 5, 6, 7, 8, 9, 10, 11].

Because of this property, the evaluation of Scattering Amplitudes, or better of the scalar form factors can be carried out in two steps: i) the decomposition in terms of MIs, usually referred to as the *reduction*, and ii) the evaluation of the MIs.

The number of MIs is sensibly smaller (even of several order of magnitude, according to the case) than the original number of scalar integrals the amplitudes depend on.

Apart from a few simple cases, the direct integration of MIs is often not viable. Neverthless, IBPs come into help, once more, and they can be used to build a system of (linear, first order) differential equations obeyed by the MIs []: therefore, solving it becomes an efficient method for the evaluation of the MIs, alternative to the direct integration techniques.

This system of differential equation for the MIs can be solved with analytical or numerical methods. Elaborating on a recent proposal that appeared in the literature [12], in this thesis we will explore a novel strategy for the numerical evaluation of MIs by means of Physics Informed Neural Network (PINN).

Machine learning, particularly deep learning [13], has permeated nearly every aspect of modern science, technology, and society. Its impact spans from advancements in computer vision and natural language processing to breakthroughs in robotics and protein folding. These techniques have introduced unprecedented capabilities for solving complex problems that were previously intractable. Recently, there has been a significant surge in applying deep learning to the numerical solution of differential equations, a development that has the potential to transform computational science, engineering and Physics.

One of the most remarkable ideas in this area is the advent of Physics-Informed Neural Networks [14, 15, 16]. PINNs are a specialized class of deep learning models that integrate physical laws directly into the learning process, making them highly effective for solving partial differential equations and ordinary differential equations. Unlike traditional machine learning models, which rely solely on data, PINNs incorporate known physical principles—typically represented as differential equations—into the model's architecture. This integration ensures that the solutions produced by the model adhere to the underlying physics of the problem, providing more accurate and physically consistent results.

The fundamental concept behind PINNs is grounded in the universal approximation capability of neural networks. Neural networks are known to approximate a wide variety of function classes, including those described by differential equations. In the PINN framework, a neural network is not merely trained to fit data; it is trained to satisfy the differential equations that govern the physical system being modeled. This is achieved by embedding the residuals of the differential equations directly into the neural network's loss function. The training process then involves minimizing this residual using gradient-based optimization techniques, driving the network toward solutions that not only fit the data but also satisfy the physical laws.

The recent study in Ref. [12] already shows promising results in solving the differential equations for MIs by PINN, and we hereby present novel ideas to ameliorate this algorithm, by further exploiting the PINN capabilities: the novel contributions we introduce in this thesis concern first the PINN implementation introducing a customizable sampling method for the training dataset and the test of a loss balancing scheme to improve the model loss function; we also propose to include more physics information in the loss function by including the higher order derivatives of the differential equation itself; moreover we also generalize the DEQ system the PINN can handle by including the possibility to have an inhomogeneous term.

1.2 Overview of the thesis

The thesis is organized as follows:

- In chapter 2 we overview the theory of Feynman integral and the strategy for their computation. In particular we show how we can shift our aim from tensorial to scalar integrals and how we can reduce their number to few, selected Master Integrals by introducing different kinds of identities. We then present how these relations can be useful to define a system of differential equations and the different methods to solve it, both analytically and numerically.

- In chapter 3 we introduce the Artificial Neural Network as a tool to solve a

great variety of tasks and outline some formal motivation to consider ANN an universal approximator in regression problems. We present the training algorithm used by a Neural Network to find the satisfied solution and explore in details the different components that can be customized for the problem at hand. Finally we point out the limitation a standard Neural Network can show even for systems as simple as an harmonic oscillator.

- In chapter 4 we introduce the Physics Informed Neural Network, explaining the differences w.r.t. a standard Neural Network in its structure and operation, we then test the improvements in performance by revisiting the harmonic oscillator problem, furthermore this study case is used to point out the differences in building a PINN compared to a standard NN. As novel contributions we further develop this study case to explore the contribution higher order PDE can provide in the learning process of PINNs and adapt a loss balancing scheme to work with the Pytorch tensor library and improve the model learning efficiency.

- In chapter 5 we further explore the PINN capabilities by addressing the Euler's hypergeometric differential equation whose solution includes the hypergeometric function ${}_2F_1$, we consider different cases including when the hypergeometric is proportional to elliptic integrals. Finally we move from a single differential equation to a system first order differential equations.

- In chapter 6 we finally return to the problem outline in chapter 2 and test a PINN model modified with our contributions in data sampling and loss balancing as a DEQ solver for the Master Integrals system of differential equations. The Feynman diagrams we consider in this chapter are both one and two-loops diagrams and their Feynman integrals can be expressed in terms of multipolylogarithms and elliptic integral as in the case of the last two examples (the equal-masses Sunrise and a non planar two-loops diagram) which we present as novel contributions of this work.

Finally in the conclusion we summarize the results of the thesis and possible extensions of this work.

2

Feynman Integrals

2.1 From Feynman diagrams to Feynman integrals

2.1.1 Tensorial decomposition

The computation of scattering of particles in quantum filed theory is typically approached in a perturbative fashion, which involves the computation of Feynman diagrams.
The description of a scattering process by Feynman diagrams is pretty straightforward: the external particle (incoming and outgoing) are represented by external legs which can then be joined by propagator to form tree patterns and loops; each vertex corresponds to the interaction term in the Lagrangian/Hamiltonian of our theory so that diagrams with an higher number of loops represents higher orders in perturbation theory.
A Feynman diagram above the tree level generally has a tensorial structure, which carries information about the nature of the particles involved and their interaction together with integrals which takes into account all possible values of the internal momenta in the loop propagators.
To deal with the integrals computation without the additional challenge of multi-

dimensional operations our first step is to factorize the tensorial part in a process known as *tensorial decomposition*: given a L-loop Feynman diagram $M^{(l)}(p)$ with E external particles with momenta p, of which n_b external bosons, this can be rewritten in the form:

$$\mathrm{M}(p) = \varepsilon_{\mu_1}(p)\varepsilon_{\mu_2}(p)...\varepsilon_{\mu_{n_b}}(p)\sum_{k=1}^{K}\mathrm{T}^{\mu_1\mu_2\cdots\mu_{n_b};k}(p)I_k(p) \tag{2.1}$$

with:

- $\varepsilon_{\mu_i}(p)$ is the polarization vector of the i-th external boson which does not depend on the internal structure of the diagram
- $\mathrm{T}^{\mu_1\mu_2\cdots\mu_{n_b};k}$ is a tensor containing information of the couplings and fermionic external legs
- $I_k(p)$, called *Form Factor* is a scalar function of the external momenta which possesses all the loops terms

This decomposition has two advantages: first it allows us to put aside the physical structure of the Feynman Diagram shifting the problem of its computation to the solution of its form factors and work only with scalar operators; second the decomposition is always possible thanks to the gauge invariance and Lorentz covariance of scattering amplitudes.

2.1.2 Feynman Integrals

All the form factors can be expressed as scalar integrals related to the original Feynman diagram: if such diagram has L loops, E external legs and I internal lines the generic expression for its scalar integral is:

$$I(p) = \int \frac{d^Dk_1}{(2\pi)^{D-2}}\frac{d^Dk_2}{(2\pi)^{D-2}}\cdots\frac{d^Dk_L}{(2\pi)^{D-2}}\frac{\Pi_{i=1}^{N_{sp}}S_i^{-n_i}}{\Pi_{j=1}^{I}D_j^{o_j}} \tag{2.2}$$

where:

- D_j corresponds to the j-th inner propagator (we will often refer to it simply as denominator), it comes from the original Feynman diagrams and is defined

as:

$$D_j = q_j^2 + m_j^2 = \left(\sum_i^{E-1} \alpha_{ji} \cdot p_i + \sum_l^{L} \beta_{jl} \cdot k_l \right)^2 + m_j^2 \tag{2.3}$$

with k_j an internal loop momentum, it's generally elevated to a power $o_j \geq 1$

- $S_i(k, p)$ in the numerator is any scalar product between either an independent external momentum and a loop momentum or by two loop momenta; for the first case we have $L\,(E-1)$ possible contractions and $L\,\frac{(L+1)}{2}$ for the latter, so the total number of products N_{sp} is:

$$N_{sp} = L\,(E-1) + L\,\frac{(L+1)}{2} = L\left(E + \frac{L}{2} - \frac{1}{2}\right) \tag{2.4}$$

  this means that N_{sp} simply depends on the number of internal propagators, but not their nature (or the one of the vertices).

However, beyond one loop the integral expression above is redundant: denominators can be repeated and typically not all scalar products in the numerators are independent, but can be re-expressed in terms of the denominators; the first step in simplifying Eq. (2.2) (the so called *trivial tensor reduction*) is then to identify the t independent denominators and express the form factors in terms of them (which will generally change their powers), secondly we re-define t of the scalar products as combinations of the independent denominator so that only $N_{isp} = N_{sp} - t$ are left, as a consequence Eq. (2.2) can be reduced to:

$$I_{o_1,\ldots,o_t,n_1,\ldots,n_{N_{\rm isp}}}(p_1,\ldots,p_{E-1}) = \int \frac{d^D k_1}{(2\pi)^{D-2}} \frac{d^D k_2}{(2\pi)^{D-2}} \cdots \frac{d^D k_L}{(2\pi)^{D-2}} \frac{\Pi_{i=1}^{N_{isp}} S_i^{-n_i}}{\Pi_{j=1}^{t} D_j^{o_j}} \tag{2.5}$$

For later convenience, we can treat the irreducible scalar product S_j as *auxiliary* denominators (raised to negative integer powers), therefore, we can introduce the

alternative definition,

$$
\begin{aligned}
I_{n_1,\ldots,n_t,n_{t+1},\ldots,n_{t+N_{\rm isp}}}(p_1,\ldots,p_{E-1}) &= \int \frac{d^Dk_1}{(2\pi)^{D-2}}\frac{d^Dk_2}{(2\pi)^{D-2}}\cdots\frac{d^Dk_L}{(2\pi)^{D-2}}\frac{1}{\Pi_{j=1}^{t}D_j^{n_j}\,\Pi_{i=1}^{N_{isp}}D_i^{n_{t+i}}} \\
&= \int \frac{d^Dk_1}{(2\pi)^{D-2}}\frac{d^Dk_2}{(2\pi)^{D-2}}\cdots\frac{d^Dk_L}{(2\pi)^{D-2}}\frac{1}{\Pi_{j=1}^{N_{sp}}D_j^{n_j}} \\
&\equiv I_{n_1,\ldots,n_{N_{\rm sp}}}(p_1,\ldots,p_{E-1}) \qquad (2.6)
\end{aligned}
$$

where we used the relation $t+N_{isp}=N_{sp}$, this notation is also used by the LiteRed program and is used for the rest of this chapter and in chapter 6. In the following, the dependence of the integral on the external momenta "$(p_1,\ldots,p_{E-1})$" will be understood, and restored when explicitly need.
This integral expression with only independent denominators and numerators is what we specifically refer to as *Feynman Integral*; this kind of reduction pursues the intent of classifying the form factors to select a limited number to actually compute, with this in mind it's useful to introduce the concept of topology.
In short, a topology, also called a Sector, is a family of scalar integrals with the same set of propagators, specifically, given the loops momenta $\mathbf{l}=\{l_i\}_{i=1,..,L}$ a topology is defined by a set of denominators $D=\{D_j=f_j(l)+m_j^2\}$ with $f_j(l)$ any function of (at least one of) the loop momenta, and a set E of independent external legs each with a given momentum and mass scale.
The definition of a topology can also be expressed as a set of rules for drawing a graph, if a denominator is associated to an oriented line with two extremal points and an external leg with an oriented line with one extremal point, these can be combined so that:

- momentum is conserved at each vertex (where two or more lines join at their extremal points)
- the resulting graph is connected

Given two topologies τ_1 and τ_2 we say that the first is a subtopology of the second if it has the same external legs and its denominators are a subset of the one of τ_2, that is:

$$D_1 \subset D_2 \quad E_1 = E_2 \qquad (2.7)$$

In the diagram representation, this means that a graph of a subtopology can be obtained from the one of its topology by shrinking one or more internal lines.
Given these definitions, it follows that a Feynman Integral as in Eq. (2.5) belongs to a topology and uniquely defines it (while usually the inverse is not true, unless the topology has only one member), moreover, considering a Feynman diagram all its form factors generate Feynman Integrals belonging to the same topology or at most its subtopologies.
The number of these Feynman Integrals can be calculated: indeed given the set $I_{t,r,s;N_{isp}}$ of all FIs with t independent propagators and N_{isp} irreducible scalar products we first define:

$$r = \sum_{j=1}^{t} n_j - 1, \quad s = -\sum_{j=1}^{N_{isp}} n_{t+j} \tag{2.8}$$

which are related to the total power of denominators and numerators respectively, then the number of FIs is:

$$N(I_{t,r,s;N_{isp}}) = \binom{r+t-1}{t-1} \binom{s+N_{isp}-1}{N_{isp}-1} \tag{2.9}$$

from this formula one can observe that for a diagram of higher order the number of Feynman Integrals associated can become quite substantial; luckily not all of them are independent and in the next section we'll explore methods to find relation among them.
As a final remark, the nomenclature can be useful to distinguish topologies: we established that if the set of propagators is known, a scalar integral can be uniquely defined by the set of their powers $I_{n_1,...,n_t,n_{t+1},...,n_{N_{isp}}}$, sometimes written as $I[\{n_1, ..., n_t, n_{t+1}, ..., n_{N_{isp}}\}]$, or shortened with just the string $\{n_1, ..., n_t, n_{t+1}, ..., n_{N_{isp}}\}$; this notation give us a quick method to determine if two integrals are in the same sector or not, for any power n in the string we replace it with 1 if $n > 0$ and 0 if $n \leq 0$, integrals in the same topology will have the same resulting string.

2.2 Identities between Scalar Integrals

2.2.1 Integration By Parts Identities

Recognizing topologies gives us an advantage to simplify calculations: since scalar integrals belonging to the same topology are not all independent, we can find relations among them and perform a recursive reduction up to a minimal set of so called *Master Integrals*; in this and the following section we'll briefly describe the most common sets of relations, starting from the Integration By Part identities.
Before addressing the details of these procedures let's consider a toy example adapted from [17], defining the class of integrals:

$$I_n(a) = \int_0^\infty dx\, x^n\, e^{-a\,x^2} \tag{2.10}$$

where $n \in N$ and $\Re[a] > 0$ so that the integrals converge, obviously computing an infinite list of integrals is an impossible task, luckily, they're not all independent; to show this we can derive an equivalence formula using the integration by part method:

$$\begin{aligned} I_n(a) &= \int_0^\infty dx\, x^n\, e^{-a\,x^2} = \\ &= \frac{x^{n+1}}{n+1}\; e^{-a\,x^2}\Big|_0^\infty - \int_0^\infty dx\; -\frac{2\,a}{n+1} x^{n+1}\; x\, e^{-a\,x^2} = \\ &= 0 + \frac{2\,a}{n+1}\int_0^\infty dx\, x^{n+2}\; e^{-a\,x^2} \\ &= \frac{2\,a}{n+1}\, I_{n+2}(a) \end{aligned} \tag{2.11}$$

after substituting $n \to n-2$ and rearranging the right and left hand side we obtain the simple relation:

$$I_n(a) = \frac{n-1}{2\,a} I_{n-2}(a) \tag{2.12}$$

for $n \geq 2$, this means that once we know the first two integrals $I_0(a)$, $I_1(a)$, the others one are given "for free" by the recurrence relation.
Observe that a completely equivalent method to find this relation between $I_n(a)$

consists in taking the derivative of their integrand:

$$0 = \int_0^\infty dx\, \frac{\partial}{\partial x}(x^n\, e^{-a\,x^2}) = \\ -2\cdot a\, \int_0^\infty dx\, x^{n+1}\, e^{-a\,x^2} + n\, \int_0^\infty dx\, x^{n-1}\, e^{-a\,x^2} \longrightarrow \\ \int_0^\infty dx\, x^n\, e^{-a\,x^2} = \frac{n-1}{2\,a} \int_0^\infty dx\, x^{n-2}\, e^{-a\,x^2} \tag{2.13}$$

where the last line is exactly the same of Eq. (2.12).
This second approach makes evident the use of the (one dimensional) Gauss Theorem, and IBPs can indeed be interpreted as its generalization of D-dimension, so for a generic scalar integral of the type defined in Eq.(2.5), we follow the same strategy by writing a vanishing integral of a divergence:

$$\int \frac{d^D k_1}{(2\pi)^{D-2}} \frac{d^D k_2}{(2\pi)^{D-2}} \cdots \frac{d^D k_L}{(2\pi)^{D-2}} \frac{\partial}{\partial k_j^\mu} \frac{u^\mu}{\Pi_{i=1}^{N_{sp}} D_i^{n_i}} = 0 \tag{2.14}$$

where k_j^μ can be any of the loop momenta ($j = 1, ..., L$) while u^μ can be any of the loop or the independent external momenta ($u \in \{l_1, ..., l_L, p_1, ..., p_{E-1}\}$).
It's natural to ask what we expect the resulting relations to be: first of all calculating explicitly the divergence would result in a linear combination of integrals and, if the integrand numerators and denominators are polynomials, we expect its coefficients to be rational functions of the kinematic invariants and the dimension variable D ; moreover, note that performing the derivative no new denominators can appear (the derivative on a denominator just raises its power, while on the numerators we just have scalar products), on the contrary the derivative of a numerator could reconstruct one if the propagator which simplify the integrand expression and if a denominator completely disappear we retrieve a member of the sub-topology.
IBP's are therefore relations, generically reading as:

$$\sum_{k=1}^{t+N_{\rm isp}} c_k\, I_{n_1,\ldots,n_k\pm 1,\ldots,n_{N_{\rm sp}}} = 0\ . \tag{2.15}$$

where the coefficients c_k are rational functions of the physical scales, namely the masses m_i's and the external invariants $p_i \cdot p_j$, and of the dimensional regularization

parameter D.
In particular, accounting for all the possibilities if we start with a scalar integral belonging to the $I_{t,s,r}$ family, IBPs identities will generate integrals in the same family, but potentially also integrals belonging to $I_{t,s-1,r}$, $I_{t,s,r+1}$, $I_{t,s+1,r+1}$ together with integrals with t - 1 propagators.

Example: As a final consideration, let's consider a simple example by computing the IBP identities for the Bubble diagram:

$$I_{n_!,n_2} = \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1^{n_1} D_2^{n_2}} \tag{2.16}$$

with

$$D_1 = k^2 - m^2 \ , \quad D_2 = (k+p)^2 - m^2 \ . \tag{2.17}$$

For simplicity, we take $n_1 = n_2 = 1$, then for the IBP we have just two possibilities:

$$\int \frac{d^D k}{(2\pi)^{D-2}} \frac{\partial}{\partial k^\mu} \frac{u^\mu}{D_1 D_2} = 0 \tag{2.18}$$

where $u^\mu \in \{p^\mu, k^\mu\}$.

- The case $u^\mu = p^\mu$

$$\begin{aligned} 0 &= \int \frac{d^D k}{(2\pi)^{D-2}} \frac{\partial}{\partial k^\mu} \frac{p^\mu}{D_1 D_2} \\ &= -\int \frac{d^D k}{(2\pi)^{D-2}} \frac{2p\cdot k}{D_1^2 D_2} - \int \frac{d^D k}{(2\pi)^{D-2}} \frac{2(p^2 + p\cdot k)}{D_1 D_2} \end{aligned} \tag{2.19}$$

we now express the scalar products in the numerators in terms of the propagators:

$$\begin{aligned} 2p\cdot k &= 2p\cdot k + k^2 + p^2 - k^2 - p^2 = D_2 - D_1 - p^2 \\ 2p\cdot k + 2p^2 &= 2p\cdot k + 2p^2 + k^2 - k^2 = D_2 - D_1 \end{aligned} \tag{2.20}$$

plugging these identities in the equation (2.19) we obtain:

$$\int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1^2 D_2} = \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1 D_2^2} \tag{2.21}$$

amounting to

$$I(2,1) = I(1,2) \ . \tag{2.22}$$

- The case $u^\mu = k^\mu$.

  This case is a bit more involved, the divergence is:

$$\begin{aligned} 0 = \int \frac{d^D k}{(2\pi)^{D-2}} \frac{\partial}{\partial k^\mu} \frac{k^\mu}{D_1 D_2} = D \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1 D_2} - \\ \int \frac{d^D k}{(2\pi)^{D-2}} \frac{2k^2}{D_1^2 D_2} - 2 \int \frac{d^D k}{(2\pi)^{D-2}} \frac{k^2 + 2p \cdot k}{D_1 D_2^2} \end{aligned} \tag{2.23}$$

  we once again express the scalar products in the numerators in terms of the propagators:

$$\begin{aligned} k^2 + 2p \cdot k = k^2 + 2p \cdot k + p^2 - m^2 - p^2 + m^2 = D_2 + m^2 - p^2 \\ k^2 = D_1 + m^2 \end{aligned} \tag{2.24}$$

  and substituting in the equation above we can simplify some of the denominators resulting in:

$$\begin{aligned} 0 = (D-3) \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1 D_2} - \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_2^2} - \\ 2m^2 \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1^2 D_2} - (2m^2 - p^2) \int \frac{d^D k}{(2\pi)^{D-2}} \frac{1}{D_1 D_2^2} \end{aligned} \tag{2.25}$$

  finally, if in the second integral we perform the linear shift $k^\mu \to k^\mu - p^\mu$, we can express the Bubble as the following linear combination of integrals, thus obtaining the second IBP relation:

$$(D-3)\, I_{1,1} = I_{2,0} + 2m^2\, I_{2,1} + (2m^2 - p^2)\, I_{1,2} \tag{2.26}$$

2.2.2 Lorentz Invariance Identities

Another common method to obtain relations between Scalar Integrals takes advantage of their nature of Lorentz scalars which implies that they should be invariants under the action of the Lorentz group O(1,3), thus for an infinitesimal Lorentz transformation the external momenta and integral transform as:

$$\begin{aligned} &p^i_\mu \to p^i_\mu + \delta p^i_\mu \qquad \delta p^i_\mu = \omega_{\mu\nu} p^i_\nu \\ &I(p^i) = I(p^i + \delta p^i) \end{aligned} \tag{2.27}$$

where $\omega_{\mu\nu} = -\omega_{\nu\mu}$ is a totally antisymmetric tensor that acts as generator of the Lorentz transformation, now we can expand the last identity:

$$I(p^i + \delta p^i) = I(p^i) + \sum_k \frac{\partial I(p^i)}{\partial p^k_\mu} \cdot \delta p^k_\mu = I(p^i) + \omega_{\mu\nu} \sum_k \frac{\partial I(p^i)}{\partial p^k_\mu} \cdot p^k_\nu \tag{2.28}$$

and if we impose the equality to $I(p^i)$ and use the antisymmetry of $\omega_{\mu\nu}$ we find:

$$\sum_j \left(p^j_\nu \frac{\partial}{\partial p^j_\mu} - p^j_\mu \frac{\partial}{\partial p^j_\nu} \right) I(p^i) = 0 \tag{2.29}$$

finally we contract the above expression with all $\frac{n(n-1)}{2}$ possible antisymmetric combination of the external momenta of the form $p^i_\mu p^j_\nu - p^i_\nu p^j_\mu = 2p^i_{[\mu} p^j_{\nu]}$ to obtain the desired identities:

$$2p^i_{[\mu} p^j_{\nu]} \cdot \sum_{k=1}^{E-1} p^k_{[\nu} \frac{\partial}{\partial p^k_{\mu]}} I(p) = 0, \quad i,j = 1, ..., E-1 \tag{2.30}$$

Note that in the last two equalities we may have been reminded of the angular momentum expression $L_{\mu\nu} = p^k_{[\mu} \frac{\partial}{\partial p^k_{\nu]}}$ and indeed Eq. (2.30) can be seen as a result of the application of the generators of the Lorentz rotations.

2.2.3 Symmetry Identities

Other identities between scalar integrals may arise by exploiting their invariance by a redefinition of the loop momenta, as explained in [18] if we take an integral

and perform a linear transformation of one (or more) of its internal momenta:

$$k_i = A_{ij} \cdot k_j + B_{ij} \cdot p_j \tag{2.31}$$

where p_j is one of the independent external momenta the value of said integral doesn't change, but its integrand will be mapped in a linear combination of other integrands and, by imposing the identity between the original and the new integral expressions, new relations can be found between different sectors or in the same sector (called *Sector Symmetries* or SecSym for short) where the set of propagators is mapped into itself.

2.2.4 Reduction to Master Integrals

Given a family of Scalar Integrals $I_{t,r,s}$ we've seen that we can produce various relations between them, from IBPs to LI and symmetry identities, but we still need to use them: let's try to take stock of the situation to understand how to proceed.

First, it's worth noting that, while for $r = s = 0$ the relations above are fewer than the integrals, as r and s increase their number grows more rapidly than the number of integrals leading to an apparently overconstrained system of equations; however the truth is that these equations are not all independent and the rank of the system will always be smaller than the number of unknowns.

This means we can't immediately solve for all the integrals, but we can still reduce them in number which in theory is trivial, but it's very time consuming; therefore a procedure with a precise order is needed, as depicted in the *Laporta algorithm* which has several code implementations.

The general idea behind it, as explained in [19], is to solve more complicated integrals in terms of "easier" ones according to an ordering defined by weights which are increasing function of the total powers sum r and s as defined in Eq. (2.8), in this way integrals with higher powers will have bigger weights; the system of equations can then be solved using Gauss' substitution rules where for each equation we isolate the integral with the highest weight and express it in term of the others, the resulting expression will then be substituted in the rest of the system. At the end of this procedure we find that all scalar integrals are represented as a linear combination of a smaller set, the so called *Master Integrals* (MIs) which

form a basis for the topology. This result dramatically reduces the number of integrals to actually compute, typically of two orders of magnitude for a two loops process: this happens because while for a topology with t propagators generally there are (t - 1) subtopologies with (t - 1) propagators, (t - 1)(t - 2) subtopologies with (t - 2) propagators and so on, some of these subtopologies can actually overlap up to be identical (as one can see with the translation of a loop momentum) so that the number of independent topologies is reduced.
Moreover since the definition of the integrals weights was arbitrary we have relative freedom in choosing which integrals to use as Master Integrals, simpler ones can be selected if we wish to solve them analytically, whereas more complicated integrals may be suitable for numerical evaluation if they show a better convergence; however for any given topology the minimum number of Master Integrals to use as a basis is fixed.
Defining as N_{M} the number of MIs, by means of IBP and LI identities, as well as of symmetry relations, generic integrals can be decomposed in terms of MIs

$$I_{n_1,\dots,n_{N_{\mathrm{sp}}}} = \sum_{i=1}^{N_{\mathrm{M}}} c_i \, M_i \tag{2.32}$$

where each master integral M_i is an integral of the type in Eq. (2.5), having a chosen set of indices n_j's.

Finally, let us remark that it has been proven in [20] that the number of MIs for a given Feynman Diagram is always finite.

2.3 Differential Equations for Master Integrals

In the previous section we found out how to build identities in order to express every integral in a topology as a combination of few Master Integrals, the last step for the analytic solution then consists in their evaluation; luckily, the same set of relations provide us a way to write a system of differential equations for the MIs which would avoid us a direct approach.
A Master Integral (like any Feynman Integral) depends on the momenta of external lines, all the non-zero mass scales and Mandelstam variable, so its derivative can be classified according to whether we derive w.r.t:

- an internal variable, i.e. an internal non-zero mass scale, this is the first method used historically [21] (Kotikov)

- an external variable chosen from the external momenta/masses or Mandelstam variables [22] (Remiddi)

we'll give an introduction to both these methods.

2.3.1 System of differential equations w.r.t. internal masses

Given a set of Master Integrals M_i, $i \in 1, ..., N_M$ let us suppose that the j-th one depends on a mass scale m_j through the denominator $(q_j^2 - m_j^2)^{-n_j}$ (and for simplicity we assume the dependence only on this denominator), we can then perform the derivative:

$$\frac{\partial}{\partial m_j^2} \int \prod_{i=1}^{L} \frac{d^D k}{(2\pi)^{D-2}} \cdots \frac{1}{(q_j^2 - m_j^2)^{n_j}} \cdots = \int \prod_{i=1}^{L} \frac{d^D k}{(2\pi)^{D-2}} \cdots \frac{-n_j}{(q_j^2 - m_j^2)^{n_j+1}} \cdots \tag{2.33}$$

the new integral has the same set of propagators and the derivative doesn't generate new numerator terms depending on momenta, therefore it belongs to the same topology and can be expressed by a combination of the MIs mentioned above, repeating this procedure for all the Master Integrals will then define a system of differential equations, calling $\vec{M}$ the vector of MIs this system can be expressed in a matrix notation :

$$\partial_{m_i^2} \vec{M} = \mathbb{A} \cdot \vec{M} \tag{2.34}$$

where $\mathbb{A}$ is the matrix associated to the system, usually called *Connection Matrix* and, since the system is obtained starting with the IBPs relations, its elements are rational functions in the kinematic invariants and the spacetime dimension D, moreover since IBPs defined identities between integrals of the same topology (and at most its subtopologies), this matrix can always be put in a block triangular form with a proper choice of Master Integrals.

2.3.2 System of differential equations w.r.t. Mandelstam invariants

Other than some mass scales, for a given physical process we can usually determine a set of kinematical invariants $\vec{s} = \{s_1, ..., s_N\}$, which suggests their use as variables for the system of differential equations; however, as shown in [23], this is not as straightforward as the mass case since the Master Integrals don't directly depends on these invariants, but rather on momenta, as an intermediate step let's then consider for a given Master Integral M the operator:

$$D_{ij} = p_i \cdot \frac{\partial M}{\partial p_j} \tag{2.35}$$

where $i, j = 1, ..., E-1$ with $E-1 = n$ the number of independent external momenta so that we can define n^2 of such operators, using the chain rule for derivative:

$$p_i \cdot \frac{\partial M}{\partial p_j} = \sum_{\alpha=1}^{N} p_i \cdot \frac{\partial s_\alpha}{\partial p_j} \frac{\partial M}{\partial s_\alpha} \tag{2.36}$$

and this can be solved to define $\frac{\partial M}{\partial s_\alpha}$ in terms of $p_i \cdot \frac{\partial M}{\partial p_j}$; note that while for the latter we can define n^2 different terms, $\frac{\partial M}{\partial s_\alpha}$ give us just $N = \frac{n(n+1)}{2}$ derivative (since the invariants can be expressed as scalar product of different momenta $s_{ij} = p_i \cdot p_j$), the set of identities above seems then overconstrained by $n^2 - \frac{n(n+1)}{2} = \frac{n(n-1)}{2}$ equations, but this is actually the exact number of LI described in section 2.2.2. Performing the derivative for all the Master Integrals will result in a system of coupled first order differential equations, which can still be represented in matricial form:

$$\partial_{s_i} \vec{\mathrm{M}} = \mathbb{A} \cdot \vec{\mathrm{M}} \tag{2.37}$$

where s_i is a kinematic invariant and the connection matrix $\mathbb{A}$ still has elements rational in $\vec{s}$ and D and can be put in a block triangular shape with the appropriate choice of MIs, that is if we start with a generic system of the Master Integrals $\vec{M}$ we can always perform a "change of basis" transformation $\vec{M} = \mathbb{T} \cdot \vec{M}'$ so that the system of differential equations becomes:

$$\frac{\partial \vec{M}'}{\partial s_i} = (\mathbb{T}^{-1}\mathbb{A}\mathbb{T} - \mathbb{T}^{-1}\frac{\partial \mathbb{T}}{\partial s_i}) \cdot \vec{M}' \tag{2.38}$$

Another common way to write the system of differential equations is by making explicit the Master Integrals $\vec{N}$ belonging to subtopologies:

$$\frac{\partial \vec{M}}{\partial s_\alpha} = \mathbb{A}_\alpha \cdot \vec{M} + \mathbb{B}_\alpha \cdot \vec{N} \tag{2.39}$$

this system then is not homogeneous for $\vec{M}$, however it can still be put in a block triangular form which suggests a bottom-up approach for its solution: we start from the simplest sectors, solving their equations and plugging the solution in the differential equations of the more complicated topologies with higher number of denominators.

2.3.3 Boundary Conditions

To define a proper Cauchy problem and find a specific solution the system of differential equation must be paired with enough boundary conditions for the MIs. Ideally the most straightforward approach would be to know the values of the Master Integrals at some points which would uniquely fix their value in a region of the phase space, however this is not always available and in that case other constraints would be necessary, for example imposing the integral to have real values in a subdomain.
Whichever method is used to determine boundary conditions, complications may arise if singularities are present: this could be actual *physical singularities* of the MIs (and consequently of the differential equations) or they may be present only in the DEs, for examples as vanishing denominators in the Connection Matrix, but the integrals are still regular.
The latter, which we call *spurious singularities* may be found checking that integrals still converge at the suspected singular points, this regularity should then be imposed since, even though these singularity are only illusory, they disconnect the MIs domain which has to be mended.
It turn out that we can actually use Eq. (2.37) to obtain relations between integrals at the spurious singularities, let's assume for simplicity a DE system on one

kinetic variable s with a spurious singularity of order n at $s = s_0$, we first rewrite the system as:

$$\frac{\partial \vec{M}}{\partial s} = \mathbb{A}_{s_0} \cdot \vec{M} + \mathbb{B} \cdot \vec{M} \tag{2.40}$$

where we decomposed the Connection Matrix so that $\mathbb{A}_{s_0}$ contain the singular terms while $\mathbb{B}$ is well behaved everywhere, then multiplying both sides for $(s-s_0)^n$ and taking the limit for $s \to s_0$ we find:

$$\begin{aligned} \lim_{s\to s_0}(s-s_0)^n \frac{\partial \vec{M}}{\partial s} &= \lim_{s\to s_0}(s-s_0)^n \left(\mathbb{A}_{s_0} \cdot \vec{M} + \mathbb{B} \cdot \vec{M}\right) \\ 0 &= \lim_{s\to s_0}(s-s_0)^n \mathbb{A}_{s_0} \cdot \vec{M} \end{aligned} \tag{2.41}$$

which will serve as a boundary condition.

2.4 Analytical method

While this Thesis focuses on a numerical techniques for the solution of Eq. (2.39), it's instructive to overview which analytic methods one can employ; first of all we just stated in section 2.3.2 how the derivation of the DEQ system by IBP identitites and the freedom in the choice of Master Integrals allow to put the system in a block triangular form which suggest a bottom-up approach, making explicit the contribution of the subtopologies the MIs system is composed by inhomogeneous differential equations, an effective technique to tackle such a problem is the *Euler's method of variation of constant*

2.4.1 Euler's method of variation of constant

Let's assume that we already solved the subtopologies so that the inhomogeneuos term in Eq. (2.39) is known and we can express it explicitly, say with a function $f(x, \varepsilon)$ so that for the i-th Master Integral we have:

$$\frac{\partial M_i}{\partial x} = \mathbb{A}_{ii} M_i + f(x, \varepsilon) \tag{2.42}$$

where for simplicity we assume that the sector we're considering has only one MI, the solution of Eq. (2.42) can be obtained by first solving the corresponding

homogeneous equation:

$$\frac{\partial M_i^{hom}}{\partial x} = \mathbb{A}_{ii} M_i^{hom} \tag{2.43}$$

which gives:

$$M_i^{hom}(x,\varepsilon) = C(\varepsilon) e^{\int^x dy \mathbb{A}_{ii}(y,\varepsilon)} \tag{2.44}$$

we can then look for a specific solution for the full equation by making an ansatz:

$$\hat{M}_i(x,\varepsilon) = \varphi(x,\varepsilon)\, M_i^{hom}(x,\varepsilon) \tag{2.45}$$

and substituting in Eq. (2.42) we obtain an equation of $\varphi(x,\varepsilon)$:

$$\frac{\partial(\varphi M_i^{hom})}{\partial x} = \mathbb{A}_{ii}(\varphi M_i^{hom}) + f(x,\varepsilon) \rightarrow \frac{\partial \varphi(x)}{\partial x} = (M_i^{hom})^{-1} f(x,\varepsilon) \tag{2.46}$$

which once solved will give us $\hat{M}_i$, then a general solution of the initial equation is provided by combination of the homogeneous and inhomogeneous ones:

$$M_i(x,\varepsilon) = M_i^{hom}(x,\varepsilon) + \hat{M}_i(x,\varepsilon) \tag{2.47}$$

2.4.2 Laurent expansion around critical dimensions

The previous method is pretty straightforward, but the dependence on ε can lead to some cumbersome calculations. Moreover we usually aim to determine the MIs in the limit $\varepsilon \rightarrow 0$ (i.e. $D \rightarrow 4$) so that the closed solution of the differential equation is often not required; therefore we can instead define the Laurent expansions of the Master Integrals:

$$M_i = \sum_{l=l_{min}}^{\infty} \varepsilon^l M_i^{(l)} \tag{2.48}$$

where l_{min} is the lowest order in the series, which we assume is a finite number since dimensional regularization divergencies are only associated to finite poles, performing the expansion also on the Connection Matrices we find for the k-th Master Integral:

$$\sum_{l=l_{min}} \varepsilon^l \frac{\partial M_k^{(l)}}{\partial s_\alpha} = \sum_{i=0} \varepsilon^i \mathbb{A}_\alpha^{(i)} \sum_{l=l_{min}} \varepsilon^l M_k^{(l)} + \sum_{j=0} \varepsilon^j \mathbb{B}_{\alpha h}^{(j)} \sum_{n=n_{min}} \varepsilon^n N_h^{(n)} \tag{2.49}$$

where n_{min} is the lowest order of the MIs belonging to the subtopologies, isolating every order in ε we obtain a chained system of differential equation that once again can be solved with a bottom-up approach:

$$\begin{aligned}
\frac{\partial M_k^{(l_{min})}}{\partial s_\alpha} &= \mathbb{A}_\alpha^{(0)} M_k^{(l_{min})} + \sum_{j=l_{min}-n} \mathbb{B}_{\alpha k}^{(j)} N_h^{(n)} \\
\frac{\partial M_k^{(l_{min}+1)}}{\partial s_\alpha} &= \mathbb{A}_\alpha^{(0)} M_k^{(l_{min}+1)} + \mathbb{A}_\alpha^{(1)} M_k^{(l_{min})} + \sum_{j=l_{min}+1-n} \mathbb{B}_{\alpha k}^{(j)} N_h^{(n)} \\
&\dots
\end{aligned} \tag{2.50}$$

this time however we don't have to take into account the dependence on ε, being factorized in the MI definition, and the DEQ system proves to be much simpler.

2.4.3 Canonical Basis

Finally, the bottom up approach is unnecessary when the differential equations system can be written in terms of a so called *Canonical Basis* [24], which means that the ε parameter can be factorized from the Connection Matrix $\mathbb{A}(\vec{x}, \varepsilon) = \varepsilon\, \mathbb{A}_{can}(\vec{x})$ so that Eq. (2.37) becomes:

$$\frac{\partial \vec{M}}{\partial x} = \varepsilon\, \mathbb{A}_{can} \cdot \vec{M} \tag{2.51}$$

for a kinematic invariant x, also in a proper canonical form $\mathbb{A}_{can}$ has at most simples poles, the equations above can be solved in an iterative way, that is using for each Master Integral their Taylor expansion in ε, it follows that the l-th order

is related to the l - 1 by:

$$\frac{\partial \vec{M}}{\partial x} = \varepsilon\, \mathbb{A}_{can} \cdot \vec{M} \rightarrow \frac{\partial \sum_l \varepsilon^l \vec{M}^{(l)}}{\partial x} = \varepsilon\, \mathbb{A}_{can} \cdot \sum_l \varepsilon^l \vec{M}^{(l)}$$

$$\rightarrow \sum_l \varepsilon^l \frac{\partial \vec{M}^{(l)}}{\partial x} = \mathbb{A}_{can} \cdot \sum_l \varepsilon^{l+1} \vec{M}^{(l)} = \mathbb{A}_{can} \cdot \sum_l \varepsilon^l \vec{M}^{(l-1)}$$

$$\varepsilon^l : \frac{\partial \vec{M}^{(l)}(x)}{\partial x} = \mathbb{A}_{can}(x) \cdot \vec{M}^{(l-1)}(x) \tag{2.52}$$

at 0-th order the equation is just a vanishing derivative, which means that the leading term is constant in the kinematic invariant:

$$\frac{\partial \vec{M}^{(0)}(x,\varepsilon)}{\partial x} = 0 \rightarrow \vec{M}^{(0)}(x,\varepsilon) = \vec{M}^{(0)}(\varepsilon) \tag{2.53}$$

at this point the entire set of Master Integrals can be defined with a Dyson Series:

$$\vec{M}(x,\varepsilon) = \left(1 + \varepsilon \int^x dy \mathbb{A}_{can}(y) + \varepsilon^2 \int^x dy_1 \int^{y_1} dy_2 \mathbb{A}_{can}(y_1)\mathbb{A}_{can}(y_2)\right) \vec{M}_0(\varepsilon) \tag{2.54}$$

which saves the trouble to deal with each Integral separately.
The canonical form is not always achievable, however it does exists a method to derive it whenever the DEQ system is linear in ε as proposed in [25]:

$$\frac{\partial \vec{M}(x,\varepsilon)}{\partial x} = (\mathbb{A}_0(x) + \varepsilon\, \mathbb{A}_1(x)) \cdot \vec{M}(x,\varepsilon) \tag{2.55}$$

Recalling that the choice of Master Integrals is arbitrary, the basic idea is to perform a "change of basis" where the projection of $\mathbb{A}_0$ vanish, in practice if we

redefine $\vec{M}(x,\varepsilon) = \mathbb{C}(x)\cdot\vec{F}(x,\varepsilon)$ equation (2.55) becomes:

$$\begin{aligned}
&\frac{\partial \mathbb{C}(x)\cdot\vec{F}(x,\varepsilon)}{\partial x} = (\mathbb{A}_0(x)+\varepsilon\,\mathbb{A}_1(x))\mathbb{C}(x)\cdot\vec{F}(x,\varepsilon) \rightarrow \\
&\frac{\partial \mathbb{C}(x)}{\partial x}\cdot\vec{F}(x,\varepsilon) + \mathbb{C}(x)\cdot\frac{\partial\vec{F}(x,\varepsilon)}{\partial x} = (\mathbb{A}_0(x)+\varepsilon\,\mathbb{A}_1(x))\mathbb{C}(x)\cdot\vec{F}(x,\varepsilon) \rightarrow \\
&\frac{\partial\vec{F}(x,\varepsilon)}{\partial x} = \mathbb{C}^{-1}(x)\left(-\frac{\partial\mathbb{C}(x)}{\partial x} + \mathbb{A}_0\mathbb{C}(x) + \varepsilon\mathbb{A}_1\mathbb{C}(x)\right)\cdot\vec{F}(x,\varepsilon)
\end{aligned} \tag{2.56}$$

it follows that if the transformation matrix satisfy:

$$\frac{\partial\mathbb{C}(x)}{\partial x} = \mathbb{A}_0\cdot\mathbb{C}(x) \tag{2.57}$$

the new set of Master Integrals obey a system in canonical form:

$$\frac{\partial\vec{F}(x,\varepsilon)}{\partial x} = \varepsilon\,\mathbb{C}^{-1}(x)\mathbb{A}_1\mathbb{C}(x)\cdot\vec{F}(x,\varepsilon) = \varepsilon\,\mathbb{A}_{can}(x)\cdot\vec{F}(x,\varepsilon) \tag{2.58}$$

with $\mathbb{A}_{can}(x) = \mathbb{C}^{-1}(x)\mathbb{A}_1\mathbb{C}(x)$, the only problem left then is to solve Eq. (2.57), the solution can once again achieved with iterated integrals of commutators:

$$\begin{aligned}
\Omega_1[\mathbb{A}_0] &= \int^x dy\mathbb{A}_0(y) \\
\Omega_2[\mathbb{A}_0] &= \int^x dy_1\int^{y_1} dy_2\,[\mathbb{A}_0(y_1),\mathbb{A}_0(y_2)] \\
&\ldots
\end{aligned} \tag{2.59}$$

the transformation matrix is just the exponential of the integral series:

$$\mathbb{C}(x) = e^{\Omega[\mathbb{A}_0(x)]},\quad \Omega[\mathbb{A}_0(x)] = \sum_{i=1}\Omega_i[\mathbb{A}_0(x)] \tag{2.60}$$

2.5 Numerical method

The definition of form factors and the reduction by IBP identities provide a significant simplification in the computation of Feynman integrals, however, the resulting Master Integrals (and their DEQ system) may still be too complicated

to deal with analytical methods; for this reasons, several numerical methods have been developed: in this section we give an overview of AMFlow, a MATHEMATICA package presented in [26] that we use in this Thesis for producing necessary data points for the Neural Network application.
The AMFLow operation is based on the the auxiliary mass flow method: we consider a Feynman integral family with t inverse propagator and N_{isp} irreducible scalar products:

$$I_{\vec{n}}(\vec{s},\varepsilon) = \int \prod_{l=1}^{L} \frac{d^D k_l}{i\pi^{\frac{D}{2}}} \frac{1}{D_1^{n_1} D_2^{n_2} ... D_t^{n_t} D_{t+1}^{n_{t+1}} ... D_{N_{sp}}^{n_{N_{sp}}}} \tag{2.61}$$

(note the different prefactor $-i\pi^{-\frac{D}{2}}$, which will be used in chapters 6 and 6.3) we first introduce an auxiliary integral family obtained from Eq. (2.61) by adding an auxiliary "mass squared" parameter η to each propagator:

$$I_{aux,\vec{n}}(\vec{s},\varepsilon) = \int \prod_{l=1}^{L} \frac{d^D k_l}{i\pi^{\frac{D}{2}}} \frac{1}{(D_1-\eta)^{n_1}(D_2-\eta)^{n_2} ... (D_t-\eta)^{n_t} D_{t+1}^{n_{t+1}} ... D_{N_{sp}}^{n_{N_{sp}}}} \tag{2.62}$$

the original integrals family can be recovered with the limit:

$$I_{\vec{n}}(\vec{s},\varepsilon) = \lim_{\eta \to i0^-} I_{aux,\vec{n}}(\vec{s},\varepsilon,\eta) \tag{2.63}$$

Auxiliary integrals can also be expressed in terms of a smaller set of MIs $\vec{M}_{aux}$, moreover the introduction of a fictious mass scale suggests a new system of differential equations:

$$\frac{\partial \vec{M}_{aux}}{\partial \eta} = \mathbb{A}_\eta \cdot \vec{M}_{aux} \tag{2.64}$$

This modification seem to introduce more complicated forms: first of all defining η_{th} as the maximum threshold for the given process the auxiliary integrals are real-valued on the real axis only for $\eta > \eta_{th}$ therefore we need to define a branch cut on the real axis from $\eta = \infty$ to η_{th}; second the number of auxiliary Master Integrals can increase w.r.t. the original integral family. However the advantage of working with integrals of Eq. (2.62) is that they greatly simplify in the neighbourhood of $\eta = \infty$, in this limit an inverse propagator with loop momentum l, external

momentum p, and mass scale m becomes:

$$\frac{1}{((l+p)^2-m^2-\eta)^n} = \frac{1}{(l^2-\eta)^2}\sum_{i=0}^{\infty}\frac{(n)_i}{i!}\left(-\frac{p^2+2p\cdot l-m^2}{l^2-\eta}\right)^i \tag{2.65}$$

as a result Eq. (2.62) turns into a combination of equal mass vacuum integrals which are much simpler than the original ones, we can then solve them and recover the original integral family by analytic continuation. The analytic continuation is performed as follow: we first define a path from $\eta=\infty$ to $\eta=i0^-$ characterized by a list of connecting points $\{\eta_1,...,\eta_N\}$ where the integrals are regular, then we proceed iteratively:

- we expand the integrals around $\eta=\infty$ and evaluate them at $\eta=\eta_1$.
- for $i=1,..,N-1$ we expand the integrals around $\eta=\eta_i$ and evaluate them at $\eta=\eta_{i+1}$.
- we expand the integrals at $\eta=0$ to match them at $\eta=\eta_N$ determining all the unknown coefficient in the asymptotic expansion.
- we take the limit for $\eta\to i0^-$.

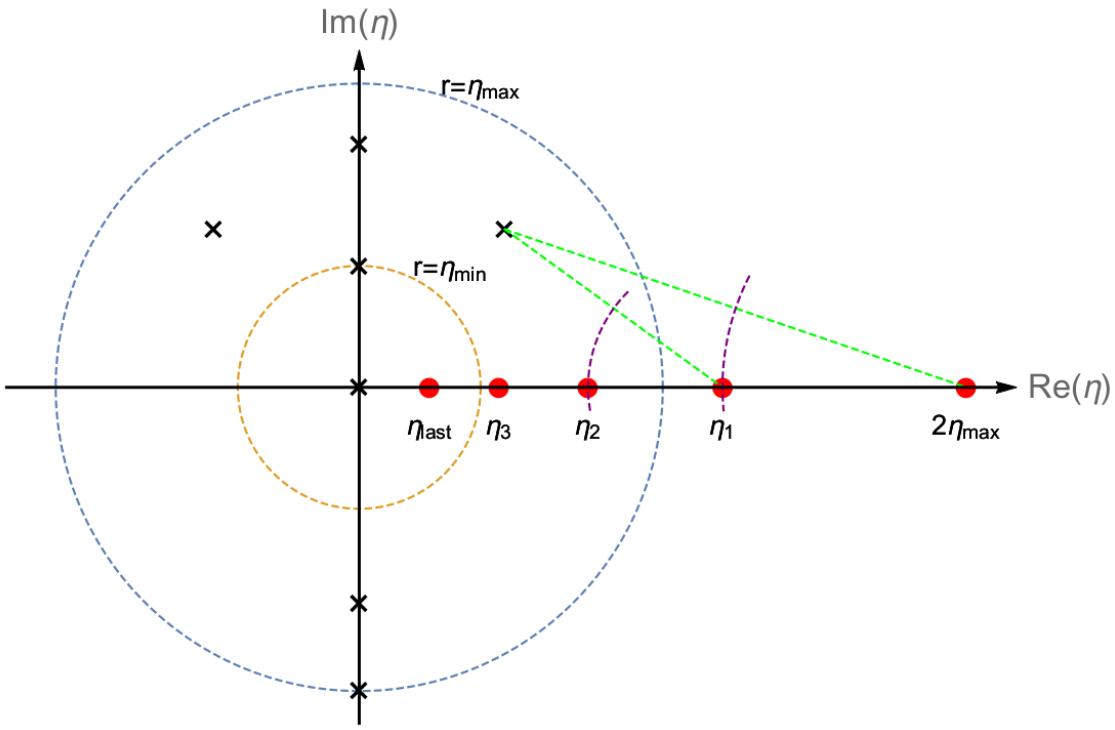


Figure 2.1: Graphical example of the analytic continuation: the singularities are represented by crosses while the red dots are the points where to expands the modified MIs, with $\eta_1 = 2\eta_{max}$ and $\eta_N = \eta_{last}$. Illustration taken from [27]

3

An introduction to Neural Networks

3.1 What is a neural network?

The use of artificial intelligence for scientific applications has been studied for the past few decades, with a significant increase of popularity in the last years with the expansion of its capabilities.
A promising approach is represented by Machine Learning (ML), a computer science field concerned with development of statistical algorithms that can learn and extrapolate from data to solve complex problems [28].
Many tasks have indeed such a level of complexity that developing a conventional algorithm to perform them would be impossible or at best impractical, ML avoid this issue by not programming the algorithm explicitly, but feeding the machine data and training it to learn and adapt to the problem at hand.
Among all the ML algorithms, one of the most popular is the Artificial Neural Network (ANN), a class of software programs whose ability to adapt is inspired by biological neural networks, i.e. animal brains: the Physics-Informed Neural Network we use in this thesis is a implementation (and an improvement) of a standard ANN, so we'll start by a general introduction on how they work.

3.1.1 Artificial Neuron

The building blocks of an ANN are the analogous to those of a brain, whose basic processing unit is the neuron: here the inputs signals collected by dendrites are added, each of them with a weight proportional to their importance; if this sum exceeds a fixed threshold the neuron is triggered and sends and signal to other neurons via synapses.
An artificial neuron is then the mathematical implementation of this structure, which is shown in Fig. 3.1: first its input is the set of features used by the model (it may be the raw data we feed into the NN, or the outputs of other neurons), and it's a collection $\vec{x}$ of n real values; these could represents anything, from the pixel values of an image to the inputs of an unknown function.
The "sum of the signals" is then performed by a linear transformation:

$$\vec{z} = \vec{\omega} \cdot \vec{x} + \vec{b} \tag{3.1}$$

where $\vec{\omega}$ is a nxn matrix of so called weights whose function is to scale differently each feature to give importance to the ones that contributes more towards the NN learning, while $\vec{b}$, called bias, is an n-vector which acts as a threshold, both weights and bias are specific for each neuron.
At this stage the neuron already gives a non-trivial output and we may ask if this structure is sufficient for regression problems (i.e. the fit of a function); with an adequately high number of neurons stacked together shouldn't we be able to approximate a function?
Actually, the capabilities of such an architecture would be very limited, since each neuron would essentially consists of a linear transformation and the composition of linear functions is still linear (whether you consider the addition of the constant represented by the bias a step beyond linearity doesn't change this limitation).
The solution to this problem is an additional transformation: the weighted sum is then passed to an *activation* function $\varphi(\vec{z})$, which corresponds to the trigger response from a real neuron and it's used to add non-linearity to the final output, so that the full mathematical expression of a neuron is:

$$NEURON_i(\vec{x}) = \varphi_i(\vec{\omega}_i \cdot \vec{x} + \vec{b}_i) \tag{3.2}$$

where we added an index identifying the neuron since a NN will typically have a large number of them, and each one has its own set of activation function, bias and weights as adjustable parameters of the network.
Nowadays there is a wide selection of activation functions to use and its choice is an important step in developing a NN, we'll give them a brief overview in the section 3.7.

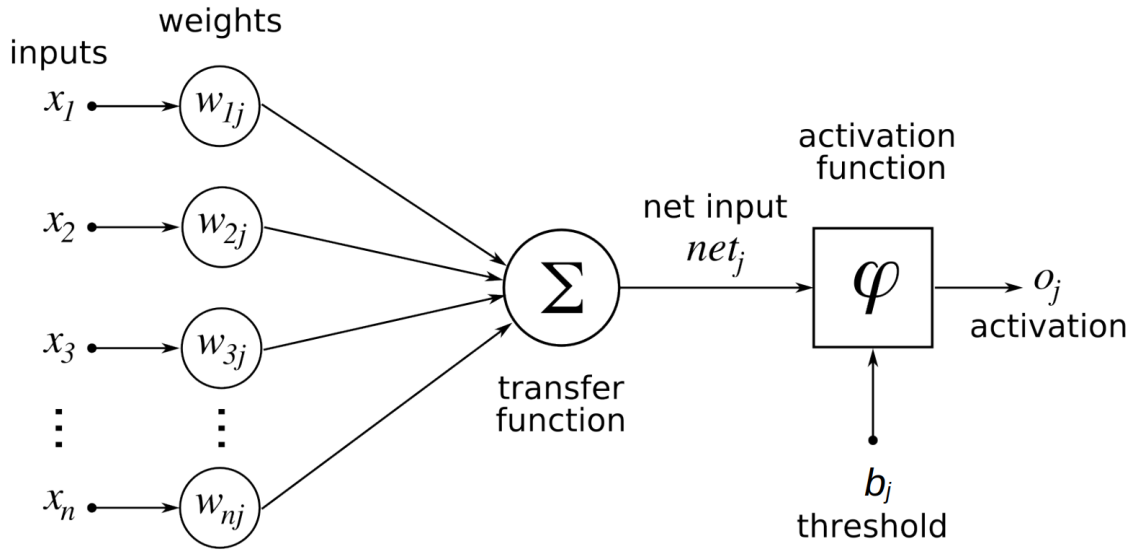


Figure 3.1: Diagram of an Artificial Neuron: each of its input $\mathbf{x_i}$ is multiplied by a weight $\boldsymbol{\omega_{ij}}$, they are all summed in the transfer function and the result together with a threshold $\mathbf{b_j}$ are mapped by an activation function $\boldsymbol{\varphi_j}$ to provide the neuron output.

3.1.2 Neural Network architecture and state

An artificial neuron by itself can be thought as the simplest example of NN architecture (historically referred to as *perceptrons*), however to perform a complex task more computational power is needed which implies the use of several neurons together, this leads to a general definition of a Neural Network:

Definition 1. *An Artificial Neural Network is a mathematical model formed by an interconnected group of units, the artificial neurons, that can learn from experience*

A Neural Network architecture is defined by its internal structure and depends on the number of neurons, the structure of connections between them and the activation functions used: a great variety of possible architectures exist, each with different capabilities.
The closest generalization of the perceptron is a series of neuron stacked in a row where the input value is passed to the first neuron, travels through the intermediate ones and in the final neuron we retrieve the output value: this architecture where the data travels only in the forward direction and there is no feedback implemented defines a so-called Feed-Forward network: even before the advent of computer this algorithm was employed in techniques like the method of least squares developed by Legendre and Gauss for data fitting.
For most applications however, this setup would still be too simple: a more common architecture is shown in Fig. 3.2 where instead of having a single neuron at each step we rather have a layer of multiple ones and the NN looks like a collection of layers: the first and the last ones are called input and output layer respectively, while all the layers in the middle are referred to as hidden, the presence of a large number of hidden layers in some models is the reason of the descriptor "Deep" in Deep Learning.
For a Neural Network to be forward oriented there's a constraint on the connections between layers, namely the neurons' outputs of one layer can only be directed to the next one, however there's some freedom in choosing how these outputs are distributed among the neurons in it: another common architecture is when the outputs of one neuron are sent to all the neurons of the next layers, in this case we refer to the Neural Network as *fully connected* and this will be the architecture used in the Thesis.

Once a NN architecture is chosen, we may ask ourselves what kind of output it will provide: to get an idea let's consider a simple model with N neurons, each one connected only to the next and let's call ω_i and b_i the weight and bias of the i-th neuron; then an input $\vec{x}$ travelling through the network will transform as

$$\begin{aligned}
\vec{x} \to \vec{y}_1 &= \varphi_1\left(\vec{\omega}_1 \cdot \vec{x} + \vec{b}_1\right) \\
\vec{y}_1 \to \vec{y}_2 &= \varphi_2\left(\vec{\omega}_2 \cdot \vec{y}_1 + \vec{b}_2\right) = \varphi_2\left(\vec{\omega}_2 \cdot \varphi_1\left(\vec{\omega}_1 \cdot \vec{x} + \vec{b}_1\right) + \vec{b}_2\right) \\
&\vdots \\
\vec{y}_{N-1} \to \vec{y}_N &= \varphi_N\left(\vec{\omega}_N \cdot \left(\varphi_{N-1}\left(...\varphi_1\left(\vec{\omega}_1 \cdot \vec{x} + \vec{b}_1\right)...\right)...\right) + \vec{b}_N\right)
\end{aligned} \tag{3.3}$$

It's pretty evident that a Neural Network output depends on the values of all the weights and biases of every neuron in its structure, more than that, once the NN architecture is set for every input $\vec{x}$ its output is *uniquely defined* by these value, we therefore define:

Definition 2. *A Neural Network model state is the collection of all the weights and biases values*

we will sometimes refer to the model state as the collection $\beta = \{\vec{\omega}_i, \vec{b}_i\}$; the way the NN will adapt and simulate the wanted solution is by repeatedly changing its model state during training.

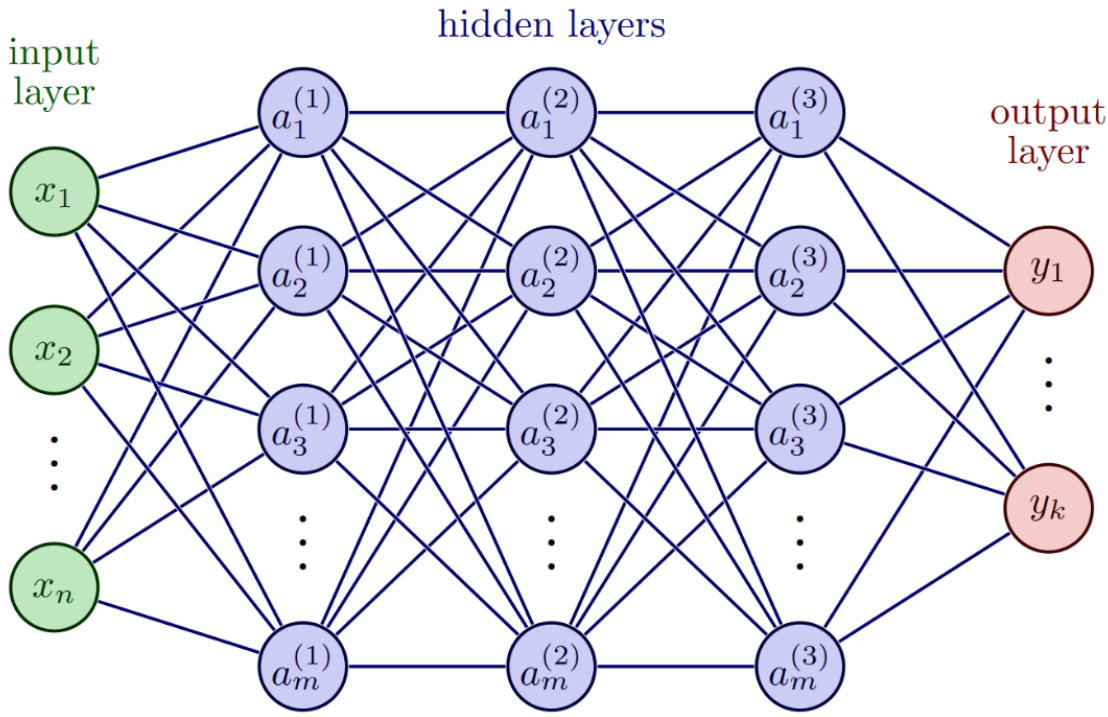


Figure 3.2: Example of a NN architecture with n input $\mathbf{x_i}$, k output $\mathbf{y_j}$ and 3 hidden layers of m neurons each. The data travel from left (input layer) to right (output layer) so the NN is feed-forward and each neuron is linked to all the neurons of the next layers, therefore the NN is fully connected. (Illustration from [29])

3.2 Universal Approximation Theorem

The presence of a large number of applications of Neural Networks testifies to their versatility, but one may wonder whether there is a formal result regarding their scope.

A rigorous treatment of a NN capabilities were carried out first by Cybenko [30] and by Hornik, Stinchcombe, and White [31] which led to the so called *Universal Approximation Theorems* that cover different cases according to the target function and NN architecture involved; one of their implication is that a Neural Network can approximate any continuous function, no matter how complicated its behaviour is, not only that, but we can restrict the NN to have just one single hidden layer (of large enough size) and the last statement still holds true; formally:

Theorem 1. *A Neural Network with just a single hidden layers can approximate at any precision order any continuous function for inputs within a specific range*

What this means is that if we set a desired accuracy ε to know a target function $\vec{y}_{true}$, it's guaranteed that exists a single layer Neural Network with a large enough number of neurons such that its output $\vec{y}_{nn}$ satisfies $|\vec{y}_{true}(\vec{x}) - \vec{y}_{nn}(\vec{x})| < \varepsilon$ for any input $\vec{x}$; the only limitations are that the domain Ω must be finite and that the target function $\vec{y}_{true}$ must be continuous, this is a consequence of the typical NN structure which is a composition of continuous functions; however if we can divide the domain in subsets $\Omega = \cup\Omega_i$ so that $\vec{y}_{true}$ is continuous in every subset, we can usually fit the function "piecewise".
A not-so-rigorous but effective representation of how the Universal Approximation Theorem holds was presented [32], here we sum up the case for the fit of a scalar function with one input: to approximate such a function let's start with a NN with an hidden layer with just two neurons as shown in Fig. 3.3:

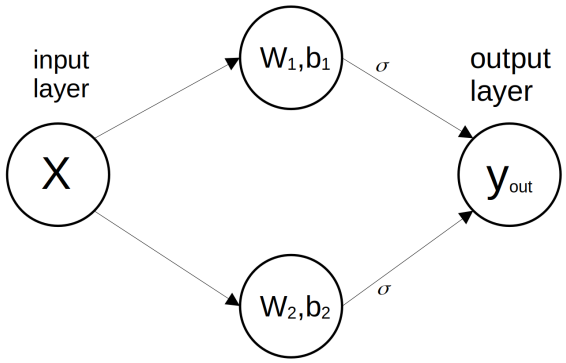


Figure 3.3: A simple Neural Network where $\mathbf{w_i}$ and $\mathbf{b_i}$ are the weights and biases of the hidden neurons and $\boldsymbol{\sigma}$ is the activation sigmoid function.

where $\sigma(z) = 1/(1 + e^{-z})$ is the sigmoid function, this is a common activation function, which has an interesting limiting behaviour as shown in Eq. (3.4): the larger the weight w of a neuron linear transformation ($z = wx + b$) is the steeper is the slope of the plot until it becomes a good approximation of the step function; the values of the bias b influences the position of the step $x_s = -b/w$, i.e:

$$\sigma(wx + b) \xrightarrow{w\to\infty} \theta\left(x + \frac{b}{w}\right) \tag{3.4}$$

An example of the sigmoid output for b = 0 and increasing values of weights is

shown in Fig. 3.4:

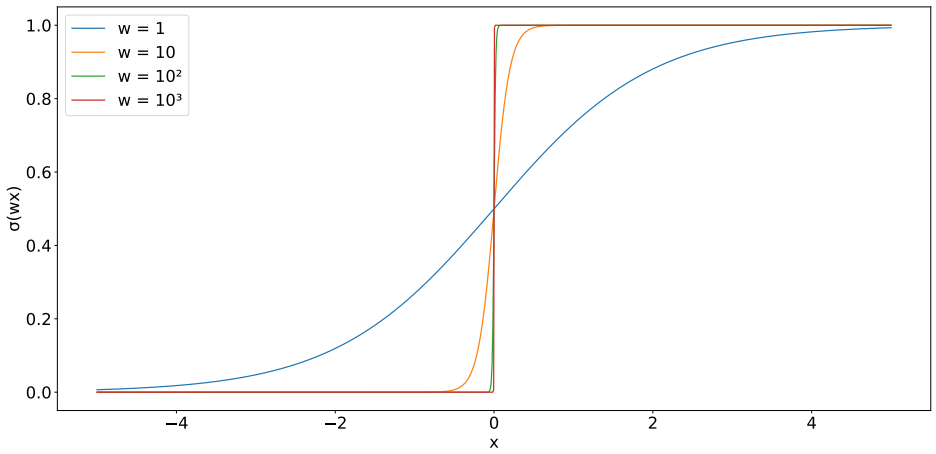


Figure 3.4: Plots of the sigmoids for b = 0 and different values of weights w.

The resulting function still have a window of failure at the position of the step since its slope will always be finite, no matter how steep, however this issue can be mended by overlapping different almost-step functions shifted w.r.t. each other, so that they cover their windows of failure.
Now let's suppose that the two hidden neurons, say neuron 1 and neuron 2, output a step-function at different position $\theta(x - x_{s1})$ and $\theta(x - x_{s2})$, respectively, with $x_{s1} < x_{s2}$. The the output neuron, say neuron 3, will receive the values of these function as inputs and, if no activation function is selected, will map them into a linear combination, generically reading as

$$y_{out} = h_1 y_1 + h_2 y_2 + b_{out} \ . \tag{3.5}$$

If we take $b_{out} = 0$ and $h_1 = h = -h_2$, the output signal of neuron 3, or simply the NN output, becomes

$$y_{out} = h(y_1 - y_2) = h(\theta(x - x_{s1}) - \theta(x - x_{s2})) \ . \tag{3.6}$$

The r.h.s. of Eq.(3.6), represents a "bump function" with x_{s1} and x_{s2} as limiting points, and h as a parameter modulating its height. An example of a NN bump

function is shown in Fig. 3.5 (left panel), for $x_{s1} = 1, x_{s1} = 2$, and $h = 1.5$. The generalization is straightforward: if we divide the function domain in N sub-intervals $[x_{min}, x_{max}] = \cup_{i=1,\dots,N}[x_{min;i}, x_{max;i}]$ we can set up a Neural Network with a layer of N pairs of hidden neurons so that the i-th neuron would approximate the function in the i-th interval, the total output of the NN will be just a multi-step plot as in Fig. 3.5 (right panel), however as N increases the approximation becomes more and more refined (the universal approximation theorem can then be derived by an application of the Stone–Weierstrass theorem).
Finally, note that while this theorem prove the existence of a Neural Network approximation for any regression problem (of a continuous function), it does not provide a way to actually find such a NN and in practice many different tools are needed to obtain passable results; in particular results like [33] show that a single hidden layer architecture is impractical to fit even simple functions.

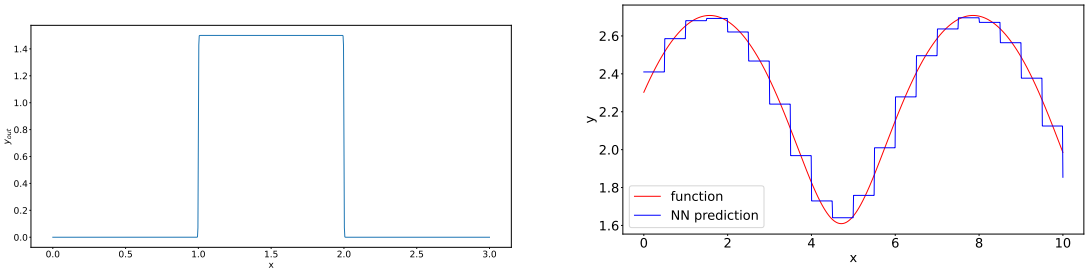


Figure 3.5: (Left panel) Example of NN bump function and (Right panel) multi step approximation of the NN for a generic function

3.3 NN training algorithm

Assuming we want to solve a regression problem for a function $\vec{y}_{true}(\vec{x})$, we can set up Neural Network choosing its architecture and initializing its state, this will gives us an output $\vec{y}_{nn}(\vec{x}, \beta)$ for any input $\vec{x}$ we feed to it, however, its values will be arbitrary since the model weights are typically set randomly or choosing a pre-defined scheme that still has no indications of our target function.
The way the NN fits the target function is what is called supervised learning, that is, a calibration of the model supported by the presence of pre-generated data $\{\vec{x}_{data}, \vec{y}_{data} = \vec{y}_{true}(\vec{x}_{data})\}$, in practice we make the model undergo a *training phase* where we compare its output with the data and modify its state β accordingly in the attempt to make their difference as little as possible (or at least under maximum tolerable level of error).
This change usually has to be done more than once to obtain satisfying result, so that during training this process is repeated several time: an outline of the training setup and loop are discussed in the next two subsections and the flowchart of the training algorithm is shown in Fig. 3.6.

3.3.1 Training Setup

Before commencing the training process, it is essential to prepare the dataset. This data may be sourced from an existing dataset or generated through other numerical evaluation methods. Additionally three core components must be configured to enable the Neural Network to optimize towards the desired solution:

- the NN model: it's the software implementation of the mathematical model described in Sec. 3.1, aimed to replicate the desired solution; its setup requires specifying its architecture, namely the number and size of layers, the activation function chosen,and the dimensions of the input and output that the model will handle.
  Moreover, its weights and biases can be initialized both randomly or according a specific scheme.

- the Loss: The loss is a function that quantifies the error between the model prediction and the actual target values.

- the optimizer: this software specifies the algorithm used to update all the weights and biases of the model in a process called backward propagation; usually a built-in optimizer provided by Pytorch is employed.
  The optimizer needs to be initialized with a set of arguments, which invariably includes the parameters to be updated (the collection β of model weights).

3.3.2 Training Loop

Once the setup is complete, the training loop can begin. This loop consists of multiple iterations, commonly referred to as epochs.
Each epoch is divided into two primary steps: the forward step and the backward step.

Forward step: In this step we "travel the NN forward" feeding into it the data to obtains its prediction and assess the NN performance, that is:

- We compute the model prediction on the training data $\vec{y}_{nn}(\vec{x}_{data})$
- we compute the loss function $L_{NN}(\vec{y}_{nn}(\vec{x}_{data}), \vec{y}_{data})$

Backward step: The information of the loss is transferred form a layer to the one before (hence the "backward" in the name), the optimizer then use this information to update the model weights and therefore, its state.

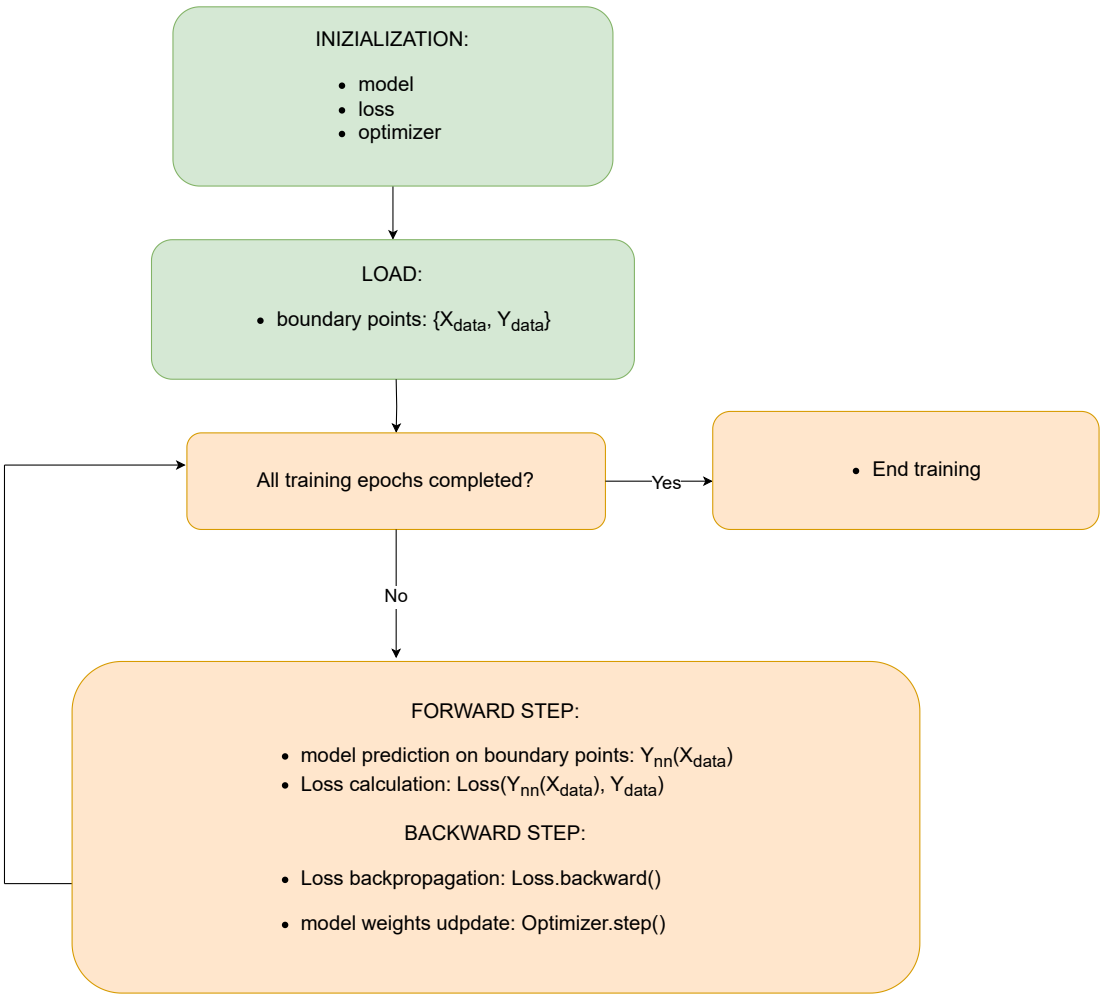


Figure 3.6: Flowchart of a NN training algorithm

The details of the loss function are discussed in Sec. 3.4 while the optimizer used throughout this thesis, called Adam, is presented in Sec. 3.6

3.4 Loss function

Most learning process include a feedback mechanism that lets you evaluate mistakes and improve your predictions, similarly a NN training requires a method to compare the NN output $\vec{y}_{nn}$ and the actual solution $\vec{y}_{true}$.
The mathematical implementation of this method is encoded by the *loss* function which generally can be any function that measures the difference between its inputs; as for any tool in the NN architecture, a vast selection of losses exist, but they can generally be grouped in three categories:

- Classification loss functions, which are used for classification problems when we have to deal with discrete values.

- Ranking loss functions estimate the relative distances between values and organize a candidate dataset in a ranking system, a typical example of its use is in face verification software.

- Regression loss functions, which deal with continuous values.

for our purposes we focus on the latter, a brief overview of the most common regression loss function is then presented.

3.4.1 Mean Absolute Error

One of the simplest example of regression loss function, the Mean Absolute Error (MAE) loss or L1 loss function computes the averaged sum on the absolute error between the predicted and target function at each data point:

$$L_{MAE}(\vec{x}_i, \beta, \vec{y}_{true}) = \frac{1}{N} \sum_{i=1}^{N} |\vec{y}_{true}(\vec{x}_i) - \vec{y}_{nn}(\vec{x}_i; \beta)| \tag{3.7}$$

this is a good choice whenever we have to handle noise and the data contain several outliers, however its ability to fit meaningful pattern is limited w.r.t. other loss functions

3.4.2 Mean Squared Error

A popular choice for regression problems, the Mean Squared Error (MSE) loss computes the square difference between the predicted and target function:

$$L_{MSE}(\vec{x}, \beta, \vec{y}_{true}) = \frac{1}{N} \sum_{i=1}^{N} |\vec{y}_{true}(\vec{x}_i) - \vec{y}_{nn}(\vec{x}_i; \beta)|^2 \tag{3.8}$$

this metric give greater emphasis to significant differences between $\vec{y}_{nn}$ and $\vec{y}_{true}$ while minor differences are penalized less, a possible drawback lies in the squaring operation since large values of $|\vec{y}_{true} - \vec{y}_{nn}|$ would be inflated and consequently impair the learning step.

3.4.3 Smooth L1 loss

This function combines the advantages of the two previous losses while mitigating their issues through an heuristic value α: if the absolute error falls below α its squared value won't be too big and it will output the MSE loss, otherwise the Mean Absolute Error will be used:

$$L_1(\vec{x}, \beta, \vec{y}_{true}) = \begin{cases} \frac{1}{N} \sum_{i=1}^{N} 0.5 \, \frac{|\vec{y}_{true}(\vec{x}_i) - \vec{y}_{nn}(\vec{x}_i; \beta)|^2}{\alpha} \\ \frac{1}{N} \sum_{i=1}^{N} 0.5 |\vec{y}_{true}(\vec{x}_i) - \vec{y}_{nn}(\vec{x}_i; \beta)| - 0.5 \, \alpha \end{cases} \tag{3.9}$$

While the correct choice of the loss function is important, a common feature they share is their dependence on the model parameters β via the predicted output $\vec{y}_{NN}(\vec{x}; \beta)$, thus the way the Neural Network is guided towards correct predictions is through the solution of a minimization problem: the best possible prediction a NN can make will correspond to the minimum of the loss, i.e. a set of values for the model state $\beta_{min} = \{\vec{\omega}_{i,min}, \vec{b}_{i,min}\}$ so that for any input $\vec{x}$:

$$L_{NN}(\vec{x}, \beta_{min}, \vec{y}_{true}) \leq L_{NN}(\vec{x}, \beta, \vec{y}_{true}), \quad \forall \beta \tag{3.10}$$

The search of minimal points of a function can be performed by studying its derivative: the backward propagation of the loss in the backward step of training

associates each NN weights with the partial derivative of the Loss w.r.t. it:

$$\{\omega_i, \quad \frac{\partial L_{NN}}{\partial \omega_i}\}, \quad \forall \omega_i \in \beta \tag{3.11}$$

The gradient is then be used by the *optimization algorithm* to update the model weights at each epoch, the simplest example is the gradient descent method:

$$\begin{aligned} \omega_i^{n+1} &= \omega_i^n - \lambda \frac{\partial L}{\partial \omega_i} \\ b_i^{n+1} &= b_i^n - \lambda \frac{\partial L}{\partial b_i} \end{aligned} \tag{3.12}$$

where $\lambda > 0$ is called learning rate; knowing that the gradient of a function is directed towards its increasing values, we aim to find a (global) minimum of the loss moving in the opposite direction and the size of this step is modulated by λ: large values of the learning rate implies a quicker descend toward a lowest point, but also increase the risk to miss the minimum by taking too big of a step, while smaller values allow a more refined search at the expense of the execution time. This method however is not without challenges: first we have to figure out an effective method of performing derivatives, which will be discussed in Sec. 3.5, then there's the risk that the algorithm doesn't point to a local and not a global minimum, which led to development of different strategies to better update the model state parameters, implemented in the code by the optimizers; the one we will use in our work, called Adam, will be explained in Sec. 3.6.

3.5 Automatic Differentiation

The common way we compute derivative numerically is through finite differences approximation, that is, for a function $f(\vec{x})$, we approximate its partial derivative w.r.t. x_i as:

$$\frac{\partial f(\vec{x})}{\partial x_i} \approx \frac{f(\vec{x} + h\vec{e}_i) - f(\vec{x})}{h} \tag{3.13}$$

where $\vec{e}_i$ is the unit direction vector along $\vec{x}_i$ and h is the step size, the smaller it is the better the approximation; however being finite there will always be a truncation error, this together with round-off errors present in numerical computation

(due to a finite number of digits we can store in our hardware) can result in low accuracy; moreover since to compute the gradient of a function for a d-dimensional input we have to compute d partial derivatives at each point, their evaluation can become resources and/or time expensive.
These drawbacks lead to the adoption of another method, called automatic differentiation: the fundamental idea behind it is that every numerical operation can be described as a combination of "elementary operations" (arithmetic, trigonometric operations, logarithmic and exponential functions) whose derivatives are known [34], and therefore the partial derivative we aim to compute can be obtained with a chain operation.
As an example, let consider a scalar function of two variables:

$$y(x_1, x_2) = x_1 log(x_2) + cos(x_1) - x_2 \tag{3.14}$$

and let's say we want to calculate its partial derivative w.r.t. x_2 at point $\{x_1 = 1, x_2 = 3\}$ (every automatic differentiation method requires to choose a fixed point): first, in what's called the forward phase, we introduce some intermediate variables z_i which represent all the operations performed in the function and calculate their values:

$$\begin{aligned}
z_{-1} &= x_1 = 1 \\
z_0 &= x_2 = 3 \\
z_1 &= z_{-1} log(z_0) = log(3) \\
z_2 &= cos(z_{-1}) = cos(1) \\
z_3 &= -z_0 = -3 \\
z_4 &= z_1 + z_2 = log(3) + cos(1) \\
z_5 &= z_4 + z_3 = log(3) + cos(1) - 3
\end{aligned} \tag{3.15}$$

then, in the reverse phase, we calculate all the different partial derivatives $\hat{z}_i = \frac{\partial y}{\partial z_i}$, observing that x_2 (z_0) is present in the definition of $z1$ and z_3 we have:

$$\frac{\partial y}{\partial z_0} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial z_0} + \frac{\partial y}{\partial z_3}\frac{\partial z_3}{\partial z_0} = \hat{z}_1\frac{\partial z_1}{\partial z_0} + \hat{z}_3\frac{\partial z_3}{\partial z_0} \tag{3.16}$$

So every partial derivatives can be represented by a combination of z_i and $\hat{z}_i$, whose values are known or the result of an elementary computation.
As previously mentioned, a Neural Network employs automatic differentiation to compute the gradient of the loss function w.r.t. the model parameters in order to update them for a better fit: the most frequently used algorithm is the *back propagation*. This algorithm happens in two step:

- the forward propagation, where an input batch of points is fed to the NN which makes a prediction of the correct output
- the backward propagation where the error on the prediction is computed and used to update the NN parameters.

In the forward pass the NN keeps a record of all the data (ingoing and outgoing) and all executed operations in a directed acyclic graph: the leaves of this graph correspond to the NN inputs, the roots correspond to the output and the nodes in between store the intermediate data.
In the backward pass the graph is traced from roots to leaves, computing the gradient at each nodes and accumulating them all the way back to the leaf corresponding to the input data we're computing the derivative.

3.6 Adam optimizer

As we have established, the way a NN learns to fit its target is by updating its weights and biases; an optimizer is a function or algorithm designed for this task and Adam (Adaptive Moment Estimation), presented in [35], is a very popular choice due to its high efficiency, especially when dealing with complicated function or highly variable gradients.
The fundamental ideas behind Adam is to adjust the learning rates for each parameter in the model: indeed when moving on the "loss landscape" (seeing the loss has a function of a multi-valued input defined by the model's state parameters β) there are directions where the loss is near-flat and it's justified to take longer step in the search for the minimum while in other directions the loss may change rapidly and it arises the need to take smaller, more cautious steps, so it's necessary to adjust each parameters' update accordingly; Adam does exactly that,

which results in better performance than having a fixed step size.
The way Adam evaluates the loss slope is by calculating its gradients, and in order to make more informed adjustments it keeps track of the gradients of the previous step and combine both the information in a weighted combination, that is, Adam combine two gradient descent methods: the Momentum method which uses the first order gradient (or first moment) and the Root Mean Squared Propagation (RMSProp) method which uses the second-order gradient (or second moment).

3.6.1 Initialization

At first, Adam initializes two vectors, $\vec{m}$ and $\vec{v}$ and a scalar t, both vectors of the same shape of the model's state parameters β, the vector $\vec{m}$ is used to store the moving average of the gradients while $\vec{v}$ is used to store the moving average of the squared gradients, the scalar t is a time step counter increased for each iteration of the algorithm, their initial values usually are:

$$\begin{aligned} \vec{m}_0 &= \vec{0} \\ \vec{v}_0 &= \vec{0} \\ t_0 &= 0 \end{aligned} \tag{3.17}$$

3.6.2 Gradients computation

At each iterations t we compute the gradient $\vec{DL}_t$ of the function we want to minimize, in our case the loss function L:

$$\vec{DL}_t = \vec{\nabla}_{\vec{\theta}} L_t(\theta_{t-1}) \tag{3.18}$$

where $\vec{\theta}_t$ represent the model's parameters values at iteration step t and $\vec{\nabla}_\theta$ is their gradient.

3.6.3 First moment update

We update the moving average of the gradients to store in the vector $\vec{m}$ as follows:

$$\vec{m}_t = (1 - \beta_1)\vec{DL}_t + \beta_1 \vec{m}_{t-1} \tag{3.19}$$

the new value is then a combination of the previous gradient and the new one, scaled by parameters β_1 and $1-\beta_1$ with β_1 a scalar called *exponential decay rate* for the first moment, usually set around 0.9 to emphasizes the last gradient; having a look-back on the past values ensures a smoother evaluation of the gradient direction to follow.

3.6.4 Second moment update

We likewise update the squared gradients $\vec{v}$:

$$\vec{v}_t = (1-\beta_2)\vec{DL}_t^{\,2} + \beta_2 \vec{v}_{t-1} \tag{3.20}$$

with β_2 the exponential decay rate for the second moment, usually set at 0.999, the second moment is an indicator of the variance of the gradients, thus signaling their reliability.

3.6.5 Update the model parameters

Once calculated the moments we can update all the model weights and biases; as said before, Adam combines the *Momentum* and *RMSProp* methods together:

$$\begin{aligned} RMSProp: \ \vec{\theta}_t &= \vec{\theta}_{t-1} - \lambda \vec{m}_t \\ Momentum: \ \vec{\theta}_t &= \vec{\theta}_{t-1} - \frac{\lambda}{\sqrt{\vec{v}_t} + \varepsilon} \\ Adam: \ \vec{\theta}_t &= \theta_{t-1} - \lambda \frac{\vec{m}_t}{\sqrt{\vec{v}_t} + \varepsilon} \end{aligned} \tag{3.21}$$

where λ is still the learning rate and ε is a small enough scalar ($\sim 1e-8$) to avoid division-by-zero.
However, it has been observed that since m and v are initialized to zero and the parameters β_1 and β_2 are close to one the following first and second moments will be biased towards zero, especially in the first few steps, therefore in the algorithm is applied a correction defined by the decay rates:

$$\hat{m}_t = \frac{\vec{m}_t}{1-\beta_1}$$
$$\hat{v}_t = \frac{\vec{v}_t}{1-\beta_2}$$
$$Adam : \vec{\theta}_t = \vec{\theta}_{t-1} - \lambda \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{3.22}$$

3.7 Activation functions

We mentioned in section 3.1.1 the importance of activation functions to make a Neural Network able to understand complex patterns; among all their contributions we recall:

- adding non linearity to the Network.
- helping in the model optimization since, during backpropagation, their derivatives define the amount by which each of the weights and biases has to be updated to minimize errors.
- helping the Neural Network to set an hierarchical representation, throughout the stack of the layers of neurons, activation functions improve the model's ability to "organize" the learning process, with the first layers learning simple feature and the following ones fine-tuning for more complex behaviour.

With time and as more scientific fields are adopting Artificial Intelligence as a tool, different activation functions were added and the choice of which one to use depends on the problem, the predicted outputs and the architecture of the Neural Network, here we give a brief overview of the most commonly used.

3.7.1 Tanh

The hyperbolic function is defined for the complex variable z as:

$$Tanh(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \tag{3.23}$$

The plot of this function is shown in 3.8 (left panel): the output is constrained in the range [-1, 1] and it's symmetric w.r.t. the origin ($z = 0$), this makes it particularly suitable with classification problems but it's often used for regression model too.
However, this function has some drawbacks: the first is the so-called "vanishing gradients" issue, that is its derivative can be very small, therefore in the backpropagation step the gradients may vanish and the training can be time consuming, the second involves the use of the exponential function which can be computationally expensive for large models.

3.7.2 ReLU

The Rectified Linear Unit function, or ReLU simply outputs its input if it's positive, or returns zero, that is:

$$ReLU(z) = max(0, z) \tag{3.24}$$

The plot of this function is shown in 3.7 (left panel): note how it fulfills the activation function role of introducing non linearity being the piecewise union of *different* linear functions.
Among the advantages of ReLU first of all there is its ability to create sparsity, which means that it can activate different subset of neurons during training, so that none of them is highly dependent of some specific feature therefore introducing randomness in the model that improve its ability to learn complex patterns; second it's very computationally efficient being a simple mathematical function.
However, its ability to bring sparsity can lead to the problem called "Dying ReLU" where some of the NN neurons are permanently set to be inactive and no longer contribute during training.

3.7.3 Leaky ReLU

The Leaky ReLU activation function was introduced as an improvement of ReLU to solve the dying neurons problem introducing a small slope for negative input values, that is:

$$LeakyReLU(z) = max(\alpha \cdot z, z) \tag{3.25}$$

where $\alpha < 1$, the non-zero ouputs for $z < 0$ indeed prevent neurons to become permanently inactive preserving the full capabilities of the NN, moreover, it retains the ReLU advantages of introducing non-linearity and being computationally efficient, its plot is shown in Fig. 3.7 (right panel).

3.7.4 GeLU

The Gaussian error Linear Unit function or GeLU is nowadays one of the most popular activation function due to its impressive performance in different fields (language/speech recognition and computer vision); its equation is given by:

$$GeLU(z) = z\,\Phi(x) = z\,\frac{1}{2}[1 + erf(\frac{z}{\sqrt{2}})] \tag{3.26}$$

where Φ is the cumulative distribution function for Gaussian distribution.
The resulting plot, shown in Fig. 3.8 (right panel), is very similar to ReLU, and indeed it shares its abilities to introducing non linearity and stochasticity, however, GeLU is actually a non-convex non-monotonic function which is not linear in the positive domain and has a curvature at all points, this may allow the model to better approximate complicated functions; the main drawback of GeLU is its computational and time consuming nature that can be partially mitigated by using its approximation with the hyperbolic tangent:

$$GeLU(z) \sim z\,\frac{1}{2}[1 + Tanh(\sqrt{\frac{2}{\pi}}(z + 0.044715\,z^3))] \tag{3.27}$$

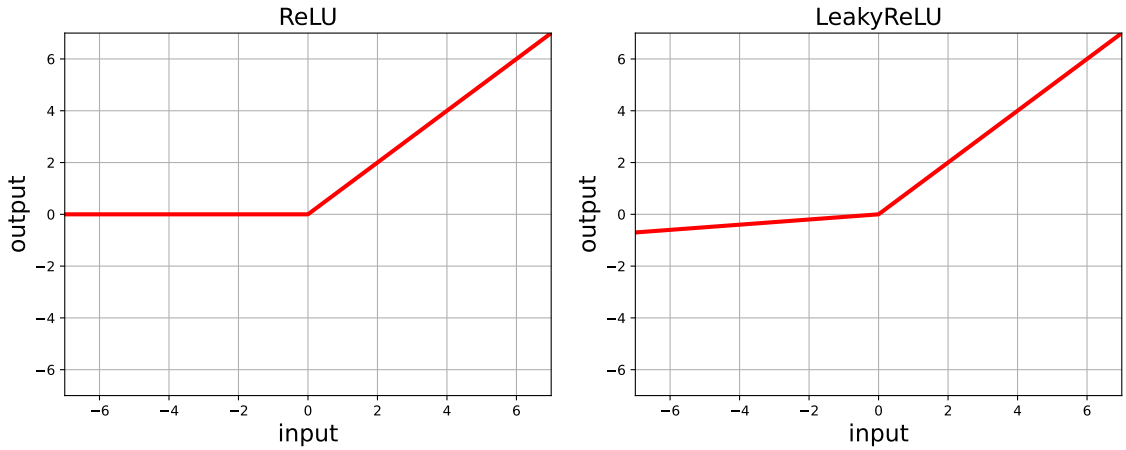


Figure 3.7: (Left panel) Plot of the ReLU activation function. (Right panel) Plot of the LeakyReLU activation function.

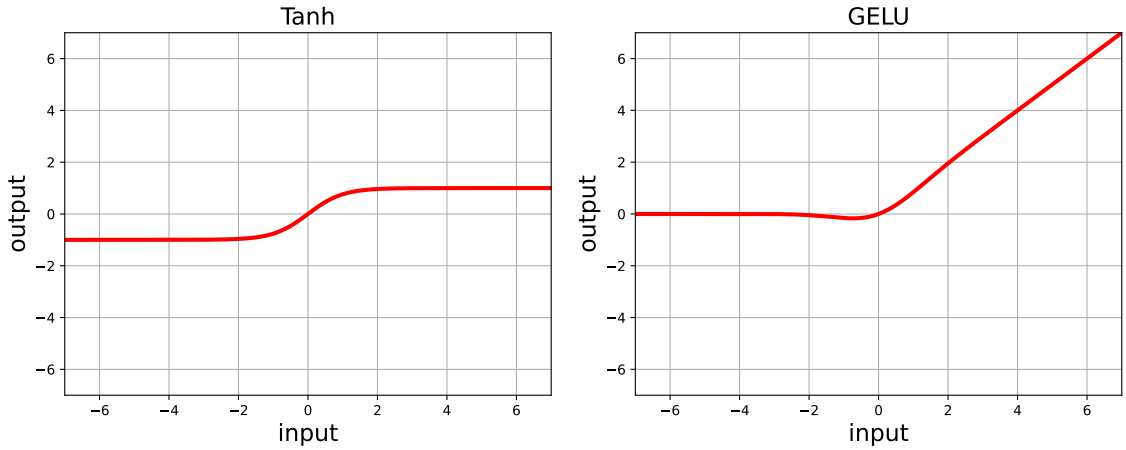


Figure 3.8: (Left panel) Plot of the Tanh activation function. (Right panel) Plot of the GELU activation function.

3.8 Limitation of a Standard Neural Network

Neural networks are powerful tools whenever a supervised learning approach is feasible, however it's not without limitations: sometimes the number of data available is very limited to ensure a satisfying fit, and in general while the model can be trained to perform effectively in the range of the independent variables used during the training a test with new data outside this range can lead to poor results.

These drawbacks can manifest even for a simple problem, as a demonstration we consider in the next section the harmonic oscillator.

3.8.1 Harmonic Oscillator

The differential equation describing a simple harmonic oscillator with angular frequency ω is:

$$\frac{d^2y(t)}{dt^2} + \omega^2 y = 0 \tag{3.28}$$

whose generic solution $y(t) = A\cos\omega t + B\sin\omega t$ is a composition of sinusoidal functions which can be fixed with two initial conditions, in our case we choose:

$$y(0) = 0, \quad \frac{dy}{dt} = 0, \tag{3.29}$$

and also consider $\omega = 20$, $t \in [0, 1]$.
The neural network architecture is minimal, with three linear layers of 32 neurons each, between them we apply the hyperbolic tangent as an activation function, which is an effective choice for regression model, the loss is simply a MSE of the NN output and the actual solution on some values of t (t_{data}); for our first fit we'll choose these data points scattered evenly in the given domain and its results are shown in the plot below:

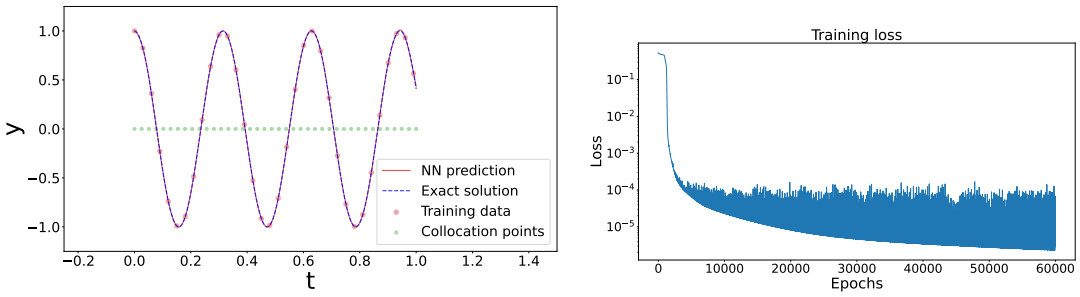


Figure 3.9: (Left panel)Harmonic oscillator solution and NN prediction after 60000 iterations. (Right panel) Loss values of the NN training for each iteration.

In this case the model is very successful in replicating the harmonic oscillator behaviour, with a mean error w.r.t. the analytic solution of order $\sim 10^{-6}$; but what would happen if the data is limited to a sub-interval instead?

To answer this question we place the data only in the first half of our domain ($t_{data} \in [0, 0.5]$) and repeat the training phase (simulating a study of a physical system where we know its the behaviour only in a certain range of the kinematic variables), the plots for the model output and the loss are shown below:

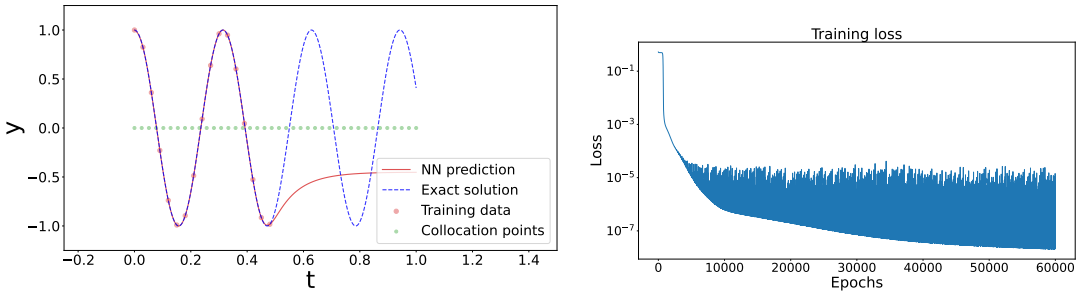


Figure 3.10: (Left panel) Harmonic oscillator solution and NN output after 60000 iterations. (Right panel) Loss values of the NN training for each iteration.

The difference is evident: while the NN represents perfectly the actual solution wherever it has data to compare to, it immediately fails once the data is absent, this issue is known as the *overfitting* of the Neural Network to the training set and seriously hinder their effectiveness to generalise outside the data range.
A brute-force way to mend to this problem was to simply increase the number of data points, which in some cases is impractical (due to the difficulty in collecting data) and ingenious tricks like batching only slightly improve the situation, a promising solution, represented by Physics Informed Neural Network, will be introduced in the next chapter.
As a final note, observe that despite the discrepancy shown in Fig. 3.10 (left panel) the loss value has a steady decreasing trend up to an order of 10^{-8} and if this was the only metric we had to check the model performance we would have a misplaced confidence in the outcome of the training; this apparent inconsistency is easily explained when you recall that loss is calculated as a mean squared loss between the model output and the actual solution *only at the data points we provide to the neural network*, that is only in the region where the NN performs adequately, while for any $t > 0.5$ the model has no indications on how well it fits the solution, and therefore on how it should update its internal parameters.
It follows that the loss used in training, while indicative, is an insufficient metric

to check the model performance, a better test would be a MSE loss calculated on points the model hasn't seen during training, that is on a set of validation points $\vec{x}_{val}$ where we know the values of the target function $\vec{y}_{val}$ we would perform the check:

$$L_{val} = \frac{1}{N_{val}} \sum_{\vec{x} \in \vec{x}_{val}} |\vec{y}_{nn}(\vec{x}) - \vec{y}_{val}|^2 \tag{3.30}$$

this testing step can be done at the end of each training iteration or at the end of training altogether.

4

Physics Informed Neural Networks

4.1 Mathematical formulation

In the previous chapter it was demonstrated how a Standard Neural Network often reveals itself to be a bad extrapolation function, that is unable to approximate the wanted result outside the training data range; the ideal solution would be something that contributes to the fit without the limitations of hard data and that can be applied at whatever scale of the independent variable; that's where Physics Informed Neural Network, introduced by Raissi, Perdikaris and Karniadakis in [[15], [16]], takes the potential of a standard ANN to another step by including the notion of the behaviour of the system in the training phase.
Whatever is the problem we aim to solve, may it be crowd flow management, structural engineering or physics dynamics, there's good chance that one or more equations that constraint its solution are known, usually in the form of a derivative equation:

$$\Xi(\vec{f},\vec{x},\vec{y}) = \sum_k f_k(\vec{x}, D(\vec{x}))\vec{y}(\vec{x}) = 0, \quad \vec{x} \in \Omega \tag{4.1}$$

where f_k are generic functions of the problem variables $\vec{x}$ and some derivative operator $D(\vec{x})$, and Ω is a domain with contour $\partial\Omega$ where we impose boundary

condition, as for example Dirichlet or Neumann condition:

$$\begin{cases} \frac{1}{N}|\vec{y}(\vec{x}) - BC|^2 & (Dirichelt) \\ \frac{1}{N}|\nabla \cdot \vec{y}(\vec{x}) - BC'|^2 & (Neumann) \end{cases} \tag{4.2}$$

for N points $\vec{x} \in \partial\Omega$ and with BC and BC' as the boundary condition for the function and its derivative respectively; it follows that if the NN output $\vec{y}_{nn}$ has to replicate the solution it would therefore need to satisfy the same equation, that is:

$$\Xi(\vec{f}, \vec{y}_{nn}, \vec{x}) = \sum_k f_k(\vec{x}, D(\vec{x}))\vec{y}_{nn}(\vec{x}) = 0 \tag{4.3}$$

the idea underlying a Physics Informed Neural Network is then to implement this equation in the learning process of the Neural Network as a form of *soft constraint* for the model, also note that the computation of Eq. 4.3 is a feasible task thanks to the use of the automatic differentiation method explained in Sec. 3.5.

4.2 Losses

If you recall, a Neural Network model computes the distance between its output $\vec{y}_{nn}$ and the true solution $\vec{y}_{true}$ in the loss function, for example using a standard Mean Squared Error:

$$L_{data}(\vec{y}_{nn}, \vec{y}_{true}) = \frac{1}{N_{data}} \sum_{\vec{x} \in \vec{x}_{data}} |\vec{y}_{nn}(\vec{x}) - \vec{y}_{true}(\vec{x})|^2 \tag{4.4}$$

and the update of the collection of NN parameters β is related to the loss derivative w.r.t. them through the gradient descent method, the loss function then seems to be the best candidate in which to integrate the physics information on the system dynamics described by the equation (4.3), we can therefore attempt to write a new term for the loss function as the residual of the PDE:

$$L_{DE} = \frac{1}{N} \sum |\Xi|^2 \tag{4.5}$$

so that the total loss would now be:

$$L_{PINN} = L_{data} + L_{DE} = \frac{1}{N_{data}} \sum_{\vec{x} \in \vec{x}_{data}} |(\vec{y}_{nn} - \vec{y}_{true})(\vec{x})|^2 + \frac{1}{N} \sum |\Xi|^2 \tag{4.6}$$

if the model output fits perfectly the actual solution, then it would satisfy the differential equation describing its behaviour therefore $L_{DE} = 0$, while for any deviation due to an imperfect fit this term would become positive penalizing the model output, therefore this new term would effectively provide a more refined metric.

Moreover, Eq. (4.3) involves only the values of the model output $\vec{y}_{nn}$ and its derivatives, while there is no dependence on data sets points, it follows that the Physics loss evaluation is not limited to the production of pre-generated data, but it can be evaluated at any point in whatever range we're interested in, these are usually called *collocation points* $\vec{x}_{coll}$ and the total loss can be more accurately defined as:

$$L_{PINN}(\vec{y}_{nn}, \vec{y}_{true}, \vec{x}_{data}, \vec{x}_{coll}) = \frac{1}{N_{data}} \sum_{\vec{x} \in \vec{x}_{data}} |(\vec{y}_{nn} - \vec{y}_{true})(\vec{x})|^2 + \frac{1}{N_{coll}} \sum_{\vec{x} \in \vec{x}_{coll}} |\Xi(\vec{y}_{nn}, \vec{x})|^2 \tag{4.7}$$

The introduction of the physics term is what promotes our model to a Physics Informed Neural Network, as a final note, it should be noted that in real examples of PINN the loss has an additional feature, that is the rescaling of its terms [36]: this is due to the fact that L_{data} and L_{DE} can (and often have) different scales, which could result in a "skewed" training phase where the model learns almost exclusively from one term and not the other, the use of a correct choice of scalar multipliers is then a way to even their contributions, the most general form of the loss is then

$$\begin{aligned} &L_{PINN}(\vec{y}_{nn}, \vec{y}_{true}, \vec{x}_{data}, \vec{x}_{coll}) = \lambda_{data} L_{data} + \lambda_{DE} L_{DE} = \\ &\lambda_{data} \frac{1}{N_{data}} \sum_{\vec{x} \in \vec{x}_{data}} |(\vec{y}_{nn} - \vec{y}_{true})(\vec{x})|^2 + \lambda_{DE} \frac{1}{N_{coll}} \sum_{\vec{x} \in \vec{x}_{coll}} |\Xi(\vec{y}_{nn}, \vec{x})|^2 \end{aligned} \tag{4.8}$$

with $\lambda_{data}, \lambda_{DE} \in \mathbb{R}^+$.

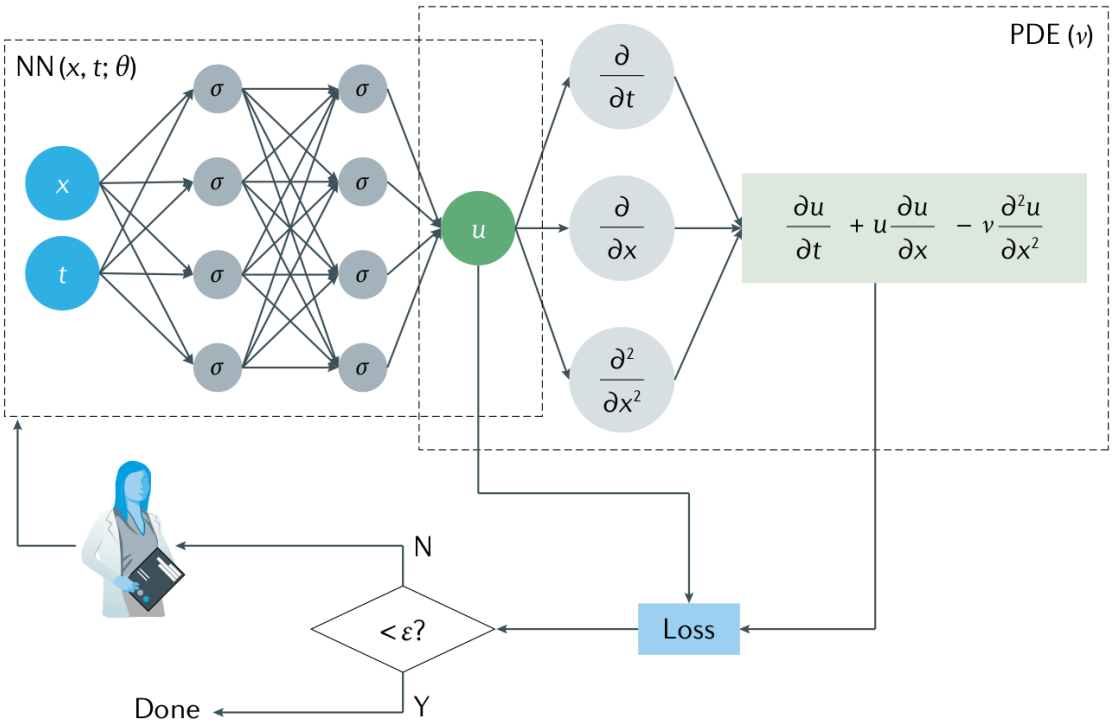


Figure 4.1: Example of a PINN model with integrated PDE (the viscous Burgers' equation), the left section of the plot represents a standard NN while the right side represents the PDE residual addition in the loss, promoting the model to a PINN. Illustration from [37]

4.3 The Loss Adaptive Balancing Scheme

In the last section we mentioned the necessity to properly scales different contribution to the loss with scalar multipliers so that they all offer equal contribution to the training phase; however this arises the important question of which values these scalars should have.

The most common approach is to just guess them, usually setting λ_{DE} some order of magnitude less than λ_{data}, since the number of collocation points is typically much higher than the number of data points; obviously this method is quite crude and could require multiple training runs on the same problem just to find the best array of λ which can be tedious; moreover, this would limit us to have a fixed set of λ values throughout training, but the balance between loss terms can change while the NN update its state.

A technique proposed by [38] aimed to solve these issues is a loss balancing scheme

called ReLoBRaLo (Relative Loss Balancing with Random Lookbacks) whose purpose is to ensure that each loss contribution improve at the same rate; this was originally implemented in TensorFlow and as a contribution for this thesis we adapted in Pytorch and integrated in the models used in chapter 5, 6 and 6.3.
The basic idea underlying this scheme is to set the scalar multiplier at each iteration proportional to the inverse of the progress of their loss terms, so that if the i-th term of the loss L_i improved just slightly in the current training step (i.e. its values is similar to the one it had previously) its corresponding multiplier λ_i will be enhanced accordingly so that it has a greater weight in the model optimization. In practice the loss term improvement at step t can be measured as the ratio between its value and the one at the beginning of training:

$$\frac{L_i(t)}{L_i(0)} \tag{4.9}$$

or comparing it to its value at the previous iteration:

$$\frac{L_i(t)}{L_i(t-1)} \tag{4.10}$$

we then need a way to set the scalars values proportionally while constraining them in a reasonable range: this is performed by the *Softmax* function which takes an array $\vec{x}$ and rescales its elements so that they lie in the range [0,1] and sum to 1:

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{4.11}$$

this would already give us all we need to implement a basic loss balancing scheme, choosing for x_i one of the ratio in Eq. (4.9) or (4.10) we can compute the new scaling as:

$$\hat{\lambda}_i(t, t') = n \frac{e^{\frac{L_i(t)}{\tau L_i(t')}}}{\sum_{j=1}^{n} e^{\frac{L_j(t)}{\tau L_j(t')}}}, \quad i = 1, \ldots, n \tag{4.12}$$

where τ is a scalar that define how much we want the loss term to improve (usually obeying $0 < \tau \leq 1$) and $t' \in \{0, t-1\}$, however this implementation has a more refined approach usually taking the improvement w.r.t. the last step, but sometimes having a "random lookback" to the beginning of training; using a Bernoulli

variable ρ we first defined an intermediate variable to choose between the two:

$$\lambda_i^{hist}(t) = \rho\,\hat{\lambda}_i(t, t-1) + (1-\rho)\,\hat{\lambda}_i(t, 0) \tag{4.13}$$

the updated values of the scalings will then be obtained by an exponential decay controlled by a second variable α (usually chosen very close to 1):

$$\lambda_i(t) = \alpha\,\lambda_i^{hist}(t) + (1-\alpha)\,\hat{\lambda}_i(t, t-1) \tag{4.14}$$

the complete procedure is then:

- Calculate the new scalings w.r.t. the previous training step ($\hat{\lambda}_i(t, t-1)$) and the beginning ($\hat{\lambda}_i(t, 0)$)
- Select the intermediate scalings λ_i^{hist} between the two of them (the one based on the relative improvement since the previous step for $\rho = 0$ or the one w.r.t. the initial step for $\rho = 1$)
- Combine $\hat{\lambda}_i(t, t-1)$ and λ_i^{hist} mediated by the parameters α: the higher its value the higher the stochasticity

4.4 PINN training algorithm

In this section we'll give an overview of the training procedure of a Physics Informed Neural Network, showcasing where it differs from a standard NN, the flowchart of the training algorithm is shown in Fig. 4.2.

4.4.1 Training Setup

Similarly to NN case, the PINN model state is updated by supervised learning therefore some data has to be prepared: the training requires at least a dataset with actual values of the target function as a boundary condition:

$$\{\vec{x}_{data}, \vec{y}_{data}\} \tag{4.15}$$

additionally, we can make an independent check on the model performance during training by comparing its prediction for a dataset the model hasn't seen:

$$\{\vec{x}_{val}, \vec{y}_{val}\}, \quad \vec{x}_{data} \cap \vec{x}_{val} = \emptyset \tag{4.16}$$

The first distinction in the setup of a Physics-Informed Neural Network (PINN) compared to a conventional Neural Network lies in the selection of collocation points $\vec{x}_{coll}$ that are necessary to compute the physics-informed term within the loss function.
It's worth noting that the collocation points do not necessarily need to be predetermined; they can be generated dynamically at the start of each epoch during the training loop. However, a method for generating these points, as well as the definition of the phase space in which they will be collocated, must be established beforehand.
Finally, we still need to set up the model, the optimizer and the loss: the procedures for setting up the model and optimizer remain unchanged from standard Neural Network training; however, the loss function in a PINN includes an additional term that evaluates the error with respect to the differential equation governing the problem, as described in Eq. (4.8). The introduction of the Physics informed term is typically tailored to the specific problem at hand, therefore the loss specifics may vary, however, loss functions for a PINN must include both data and collocation points as input:

$$L_{PINN} = L_{PINN}(\vec{y}_{data}, \vec{y}_{nn}(x_{data}), \vec{x}_{coll}, \vec{y}_{nn}(\vec{x}_{coll}), ...) \tag{4.17}$$

4.4.2 Training

Similarly to the NN case described in Sec. 3.3 each epoch is divided in a forward step and a backward step.

Forward step: This step differs from the NN case, due to the presence of the collocation points, in particular:

- We split the collocation points in different subsets called *batches*: this practice is not necessary, but it avoids to load the whole set of collocation points

to memory at once and, by randomly shuffling point into different batches at each iteration, it adds stochasticity to the PINN learning.

$$\vec{x}_{coll} = \cup\{\vec{x}_{batch}\} \tag{4.18}$$

- while looping over all the batches, we compute the model prediction for the boundary and collocation data:

$$\vec{y}_{nn}(\vec{x}_{data}), \quad \vec{y}_{nn}(\vec{x}_{batch}) \tag{4.19}$$

- the model prediction and the boundary data are used to compute the loss function:

$$L_{PINN}\,(\vec{y}_{nn}(\vec{x}_{data}), \vec{y}_{data}, \vec{x}_{batch}, \vec{y}_{nn}(\vec{x}_{batch})) \tag{4.20}$$

Backward step: The information on the physics is already embedded in the loss function so this step is the same as for the standard NN, that is:

- The gradients of the loss function with respect to the weights and biases of the PINN model are computed during backpropagation
- The optimizer uses the computed gradients to update the model parameters according to the algorithm selected.

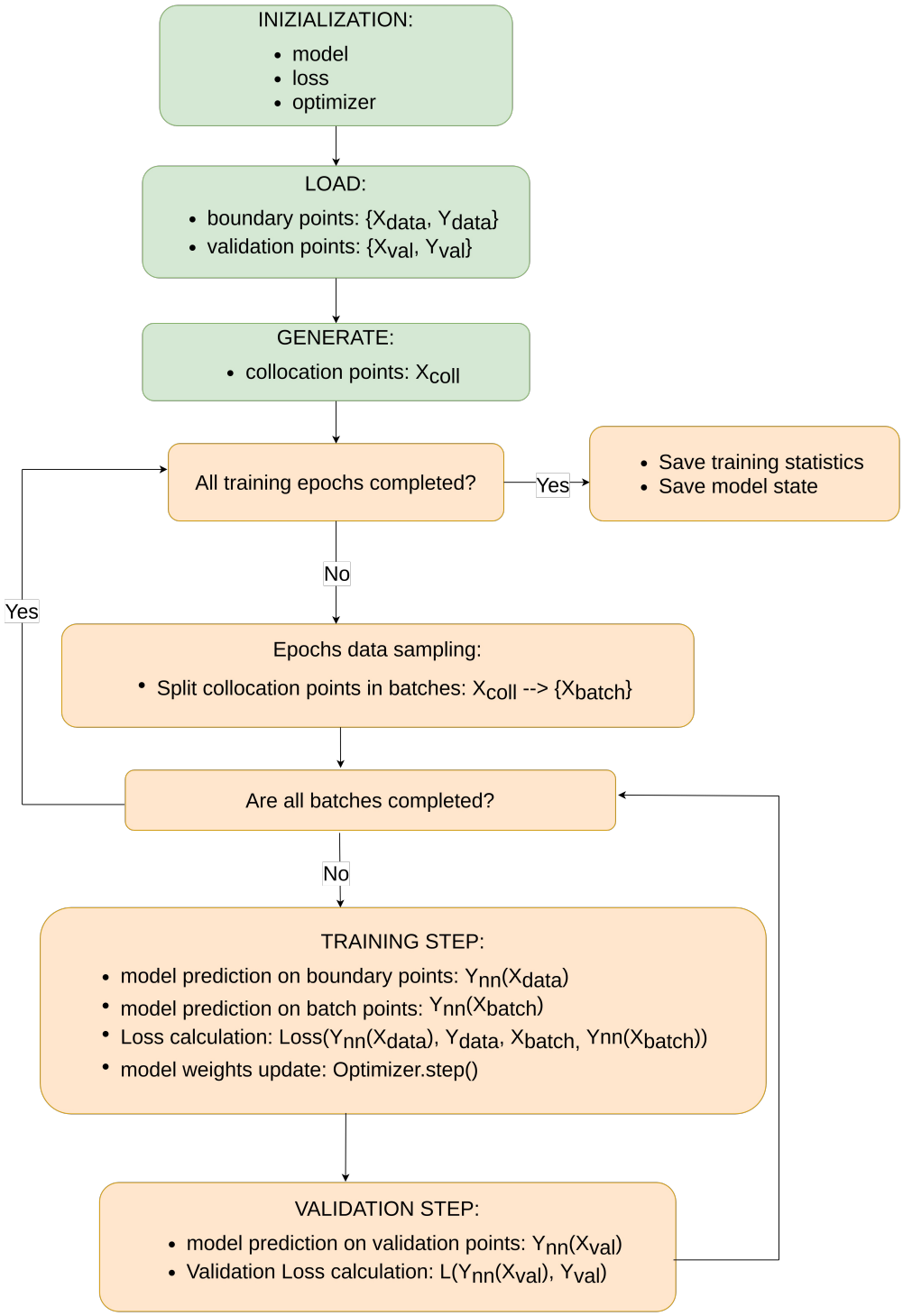


Figure 4.2: Flowchart of a PINN training algorithm

4.5 Harmonic oscillator revisited

In Sec.3.8.1 we studied an harmonic oscillator with a Standard Neural Network, and we found out its inability to extrapolate past the data, since a PINN supposedly should eliminate these limitation let's test if its implementation can lead to a better fit: we make a study similar to the one in [39], considering again $t_{data} \in [0, 0.5]$, but now we also add 40 collocations points spread evenly in the full range [0, 1], the harmonic differential equation (3.28) will define a term of the physical loss, but that's not the only equation related to our system; in fact we also know the equation for its energy:

$$\frac{1}{2}\left(\frac{dy}{dt}\right)^2 + \omega^2 y^2 = E = \frac{\omega^2}{2} \tag{4.21}$$

the left hand side can be evaluated for the NN output (substituting $y \longrightarrow y_{nn}$) while the right hand side is set once ω is known, leading to following form of the physical loss:

$$\begin{aligned} L_{de} &= \frac{d^2 y_{nn}(t)}{dt^2} + \omega^2 y_{nn} \\ L_E &= \frac{1}{2}\left(\frac{dy_{nn}}{dt}\right)^2 + \omega^2 y_{nn}^2 \\ L_{DE} &= \lambda_{de} L_{de} + \lambda_E L_E \end{aligned} \tag{4.22}$$

The results of the training are shown in the plots below:

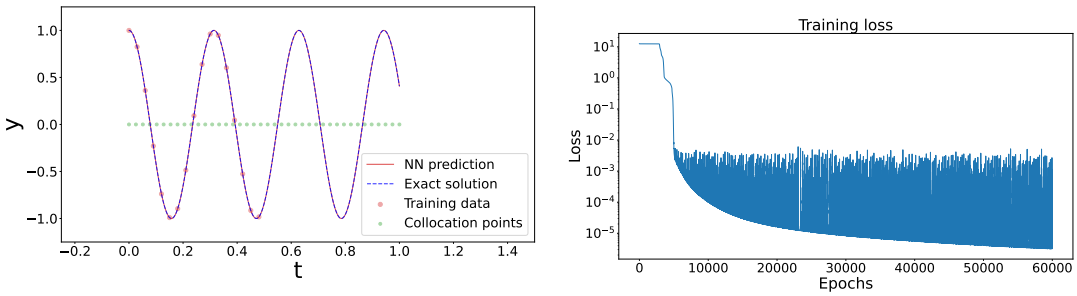


Figure 4.3: (Left panel) Harmonic oscillator solution and PINN prediction after 60000 iterations. (Right panel) Loss values of the PINN training for each iteration.

Now the PINN is able to fit the harmonic oscillator function even where direct data is absent, with a final value of the training loss of order 10^{-6} which is satisfying result for this very simple model set up; however, someone may not be convinced of the effectiveness of a Physics Informed framework from the fit of an elementary periodic functions.
A possible objection of this result is that the precision achieved is merely a consequence of the high number of data points selected in the range $[0, 0.5]$, so that the behaviour of the function is modeled by a standard data-driven optimization and the physics loss contribution just "guided" the NN output for the rest of the domain; to get to the bottom of the matter let's take the extreme case of using just one data points and train the model (with the other hyperparameters unchanged), as you can see in figure 4.4 (left panel) the model manage to accurately reproduce the shape of the harmonic function, but its output is out of phase.
Anyone familiar with differential equations theory would recognize this behaviour: recall that the physical loss term is constructed from the energy conservation law and the harmonic differential equation which are respectively a first and second order differential equation, with only one boundary condition (whose role is in our case embodied by the single data point) we are not able to completely fix the solution for both of them, to be specific is enough to fix the energy value (and therefore the frequency), but not the phase; the addition of a second data points eliminates the ambiguity as seen in Fig. 4.4 (right panel).

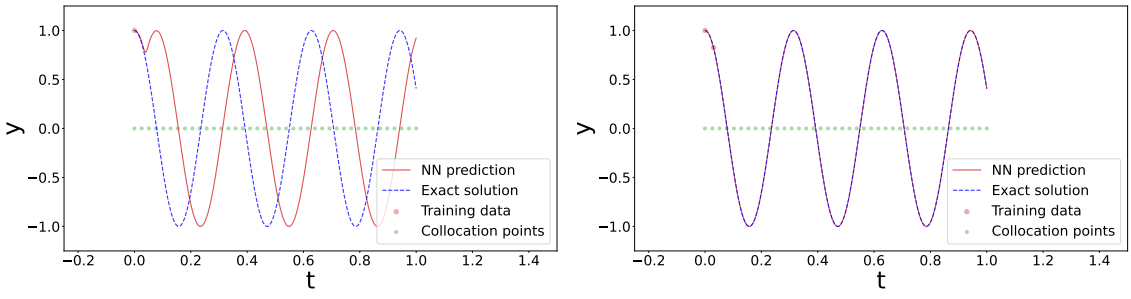


Figure 4.4: (Left panel) Harmonic oscillator solution and PINN prediction with one data point. (Right panel) Harmonic oscillator solution and PINN prediction with two data point.

4.6 Activation function for PINN

The simple example of the harmonic oscillator let us illustrate another key difference of the training procedure of a Physics Informed Neural Network with respect to a Standard one defined by the choice of activation function.
We already mentioned how there are different choice of activation function, each one is more suited to a particular task than others: broadly speaking, function with a finite number of outputs values are used in classification/binary problems, while regression usually needs a function with real values $f : \mathbb{R} \to \mathbb{R}$; however the additional terms in the PINN loss require more restrictive criteria.
To show this, let's come back to the harmonic oscillator, but instead of using the hyperbolic tangent as an activation we adopt the "Rectified Linear Unit" function, i.e. $ReLU(x) = max(0, x)$, shown in Fig. 3.7 (left panel), while leaving all other hyperparameters unchanged (size of hidden layers, number of collocation points etc.) after the same number of training iteration the result is shown in Fig. 4.5 (left panel). It's obvious that the model is now unable to fit the target y_{haos} where the data points are absent and it generally provides a rough approximation. The modified activation function is the only suspect for this result: a first possible explanation comes from the difference between the codomain of $y_{haos}(t)$ and ReLU, the latter can only assume non negative values and the same holds for the output of the any of the hidden layer; the oscillator displacement is instead both positive and negative.
We therefore make a second attempt at fitting the harmonic oscillator using an improved version of ReLU, called LeakyRelu which instead of suppressing negative inputs it maps them with a negative slope, as shown in Fig. 3.7 (right panel), the training result are shown in Fig. 4.5 (right panel). The fit has improved, but still fails to reproduce the target, especially in the range without data points: the source of the problem indeed resides in the "Physics Informed" portion of training, and specifically in the loss; L_{DE} implies derivatives up to the second order of the NN output, but both ReLU and LeakyReLU, being essentially a composition of linear function, have vanishing second derivative and therefore the corresponding physical terms vanish, leading to a non-existent (or even misleading) contribution of L_{DE} at the gradient descent optimization of the model.
Our original fit in Sec. 4.5, on the other hand, used the hyperbolic tangent which

has a non trivial second derivative: this demonstrate the need for an extra rule in designing the architecture of a PINN, i.e. to choose activation function whose differentiability class is of higher or at least of the same degree of the one of the target function.

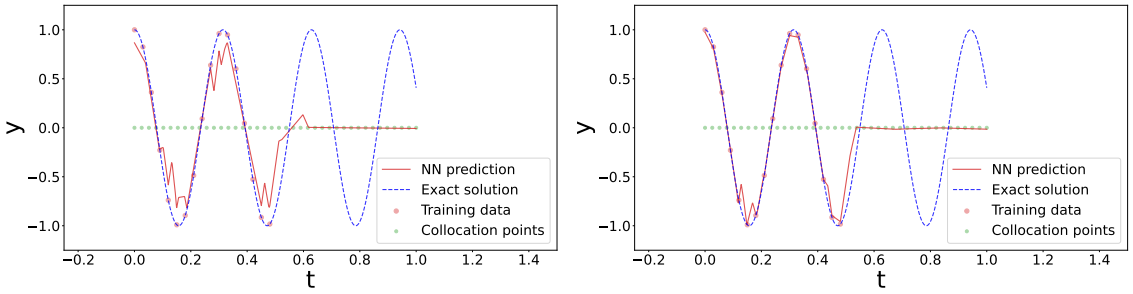


Figure 4.5: Fit of the PINN model with a ReLU activation function (left panel) and LeakyReLU activation function (right panel).

4.7 Higher order loss terms

The introduction of a physics informed term in the loss function is revealed to be an essential contribution in NN performance when data is scarce, but as seen in Sec. 4.5 we're not limited to add just one equation: the more information we have on the system dynamic the more the neural network can be guided towards the desired solution.
The harmonic oscillator can still be useful to showcase this remark: we already introduced in the previous section two Physics informed loss terms derived by the harmonic oscillator equation and the energy definition:

$$\frac{d^2 y_{nn}(t)}{dt^2} + \omega^2 y_{nn} = 0 \tag{4.23}$$

$$\frac{1}{2}\left(\frac{dy_{nn}}{dt}\right)^2 + \frac{1}{2}\omega^2 y_{nn}^2 = E_{val} \tag{4.24}$$

For this check we also include an additional term: obviously a new, independent equation would be preferable, but we want to test if even an equation derived from the ones above still brings an improvement: deriving Eq. (4.24) gives Eq.

(4.23) while the derivative of the latter results in:

$$\frac{d^3 y_{nn}(t)}{dt^3} + \omega^2 \frac{dy_{nn}(t)}{dt} = 0 \tag{4.25}$$

it follows that the physics informed loss has up to three contributions:

$$\begin{aligned} L_1 &= \frac{1}{N_{coll}} \sum_{x \in \vec{x}_{coll}} \left| \frac{d^2 y_{nn}}{dt^2} + \omega^2 y_{nn} \right|^2 \\ L_2 &= \frac{1}{N_{coll}} \sum_{x \in \vec{x}_{coll}} \left| \frac{1}{2} \left(\frac{dy_{nn}}{dt} \right)^2 + \frac{1}{2} \omega^2 y_{nn}^2 - E_{val} \right|^2 \\ L_3 &= \frac{1}{N_{coll}} \sum_{x \in \vec{x}_{coll}} \left| \frac{d^3 y_{nn}(t)}{dt^3} + \omega^2 \frac{dy_{nn}(t)}{dt} \right|^2 \end{aligned} \tag{4.26}$$

we then perform three training runs: we use the same NN architecture with 3 layers of 32 neurons each, using the hyperbolic tangent as an activation function and the model is initialized with the same weights values; the data is also the same and this time we add a validation set:

$$\begin{aligned} N_{epoch} &= 3 \times 10^4, \\ x &\in [0, 1], \\ t_{data} &\in [0, 0.1], \quad N_{data} = 3 \\ t_{coll} &\in [0, 1], \quad N_{coll} = 30 \\ t_{val} &\in [0, 1], \quad N_{val} = 10^3 \end{aligned} \tag{4.27}$$

the difference between each training is in the loss:

$$\begin{aligned} Training1: \quad & L_{DE}^1 = \lambda_1 L_1 \\ Training2: \quad & L_{DE}^2 = \lambda_1 L_1 + \lambda_2 L_2 \\ Training3: \quad & L_{DE}^3 = \lambda_1 L_1 + \lambda_2 L_2 + \lambda_3 L_3 \\ & L_{PINN}^i = \lambda_{data} L_{data} + L_{DE}^i \end{aligned} \tag{4.28}$$

we also refer to the validation loss for each run as L_{val}^i with $i = 1, 2, 3$. In Fig. 4.6 we plot the values of the validation loss at each epoch for the three training runs:

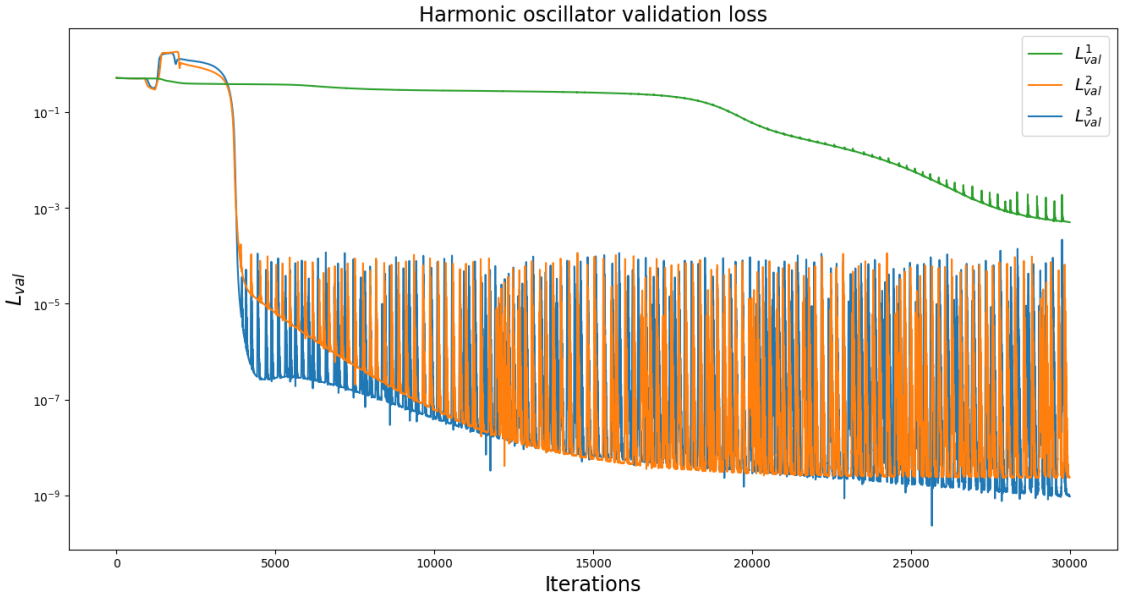


Figure 4.6: Validation loss values for different physics informed loss functions.

While the harmonic oscillator DEQ is sufficient to ensure the fit, with a validation loss of $\mathcal{O}(10^{-4})$ at the end of training, clearly the addition of the energy equation defines a significant improvement $L_{val}^2 \approx \mathcal{O}(10^{-9})$.
Finally, even though the addition of the higher order derivative loss L_3 is not as significant it still bring improvements: first of all there is a faster gain in model performance, second the final value of the validation loss is the best of the three runs with an order of $\mathcal{O}(10^{-10})$. We conclude that the use of higher order differential equations can improve the performance of a Physics informed Neural Network; however this strategy is not always viable: it was shown in previous works the limitations of PINN in dealing with advections-dominated PDEs ([40]) and chaotic equations ([41]), moreover while the introduction of new loss terms can contribute to the optimization process in the search of the loss global minimum, it can hinder it as well by creating local minima, it follows that a more populated Physics informed loss requires a careful scalar linearisation.

5

Hypergeometric functions and PINN

The goal of this chapter is to implement the concepts of the PINN method to solve the hypergeometric differential equation, whose solution is the ${}_2F_1$ hypergeometric function. The hypergeometric differential equation is a second-order linear ordinary differential equation, and its solutions admits a series as well as an integral reresentation. Hypergeometric functions encompass many special mathematical functions, such as Logarithms, Beta functions, Bessel functions, Legendre functions, and elliptic integrals, all of which frequently appear in physics. These functions can be seen as specific limiting cases of hypergeometric functions. Additionally, the connection between hypergeometric functions and Feynman integrals is significant, playing a vital role in the computational toolkit used in quantum field theory. Due to these extensive applications, hypergeometric functions provide an ideal framework to explore the efficacy of the PINN method in addressing various problems in physics. In this chapter, we show the application of PINNs to solve the hypergeometric differential equation, offering insights into its broader applicability in physics. The chapter is organized as follows: in Sec. 5.1, we describe the hypergeometric function and its different properties. In Sec. 5.2, we outline the PINN model used to solve the hypergeometric differential equation and the training results are summarized in Sec. 5.3. In Sec. 5.4, we introduce a coupled

system of first order differential equation for the hypergeometric functions and employ the PINN to study its solution.

5.1 Mathematical Formulation

The hypergeometric differential equation is a second order linear ordinary differential equation, which can be expressed as:

$$\Xi(a,b,c;x) = x(1-x)\frac{d^2f}{dx^2} + (c-(a+b+1)x)\frac{df}{dx} - abf = 0. \tag{5.1}$$

This differential equation has three regular singular points at 0, 1 and ∞ and any second order differential equation with three regular singular points can be transformed into it.

There are two linearly independent solutions for the hypergeometric differential equation and the solutions are constructed using the Gauss hypergeometric function ${}_2F_1(a,b,c,z)$. The Gauss hypergeometric function is a special function defined for the complex variable $|z| < 1$ as a power series expansion:

$${}_2F_1(a,b,c,z) = \sum_0^\infty \frac{(a)_n (b)_n}{(c)_n}\frac{z^n}{n!}, \tag{5.2}$$

where, $(p)_n$ is the rising Pochhammer symbol. It is defined as

$$(p)_n = \begin{cases} 1 & n = 0 \\ p(p+1)...(p+n-1) & n > 0, \end{cases} \tag{5.3}$$

for $|z| \geq 1$. The function can be analytically continued in all the complex plane with the exceptions of its branch point at 1 and ∞; So, the computer implementations of the hypergeometric function usually have a branch cut discontinuity for $z = Re(z) \geq 1$ like the one in MATHEMATICA. Several special mathematical functions can be obtained as the limiting cases of the hypergeometric function. For example, choosing the parameters as $a = 1$ and $b = c$, the ${}_2F_1$ reduces to the geometric series:

$${}_2F_1(1,b,b,z) = 1 + z + z^2 + z^3 + ... \ . \tag{5.4}$$

When a or b is negative then the series is truncated and the hypergeometric function becomes a polynomial:

$$ {}_2F_1(-|a|, b, c, z) = \sum_{n=0}^{|a|} (-1)^n \binom{|a|}{n} \frac{(b)_n}{(c)_n} z^n \tag{5.5} $$

The Kummer's function ($M(a, c, z)$) can also be expressed as a special limiting case of the ${}_2F_1$:

$$ M(a, c, z) = \lim_{b\to\infty} {}_2F_1(a, b, c, b^{-1} z) \tag{5.6} $$

Similarly, it can be shown that orthogonal polynomials, incomplete beta functions, Bessel functions, Legendre functions, which have several applications in Physics, including Quantum Mechanics and QFT can be expressed as limiting cases for the hypergeometric function.
Finally, a case of particular interest for our study concerns elliptic integrals: these functions are gaining importance in modern Feynman calculus as they appear in the computation of multi-loop Feynman integrals with massive, propagating particles. The elliptic integrals can also be expressed in terms of the hypergeometric function for specific choices of $\{a, b, c\}$. The complete elliptic integrals of the first and second kind can be defined as follows:

$$ \begin{aligned} K(x) &= \int_0^{\frac{\pi}{2}} \frac{dt}{\sqrt{1 - x^2 \sin t^2}}, \\ E(x) &= \int_0^{\frac{\pi}{2}} dt \sqrt{1 - x^2 \sin t^2}, \end{aligned} \tag{5.7} $$

with $x \in [0, 1)$. Now, we can expand the integrands by using the binomial theorem and employing the following identities:

$$ \begin{aligned} &\int_0^{\frac{\pi}{2}} \sin t^{2k} dt = \int_0^{\frac{\pi}{2}} \cos t^{2k} dt = \frac{\pi}{2} \frac{(2k)!}{(2^k k!)^2}, \\ &\int_0^{\frac{\pi}{2}} \sin t^{2k+1} dt = \int_0^{\frac{\pi}{2}} \cos t^{2k+1} dt = \frac{(2^k k!)^2}{(2k+1)!}, \quad \text{where, } k \in N^0, \end{aligned} \tag{5.8} $$

we can obtain a series in the form of Eq.(5.2). As a result, these elliptic integrals

are defined as

$$K(x) = \frac{\pi}{2}\,{}_2F_1\left(\frac{1}{2}, \frac{1}{2}, 1, x^2\right) \tag{5.9}$$

$$E(x) = \frac{\pi}{2}\,{}_2F_1\left(\frac{1}{2}, -\frac{1}{2}, 1, x^2\right) \tag{5.10}$$

So, the hypergeometric function is both of physical interest and very practical since we can just change the values of the parameters a, b, c to study different functions.

5.2 PINN model setup

We use a fully connected network with 4 hidden layers with 64 neurons each; both the input and the output layer have just one neuron since we limit our work to real values so both the inputs and the outputs are real scalars.
The choice of the data samples is dictated by the specific values of $\{a, b, c\}$, however being the hypergeometric differential equations of second order as a boundary conditions for the model we choose samples for both the function and its first time derivative:

$$BC \begin{cases} \{x^i_{data}, y^i_{data}\}, & i = 1, ..., N_{data} \\ \{x^i_{der}, y^i_{der}\}, & i = 1, ..., N_{der} \end{cases} \tag{5.11}$$

this data is generated using the analytical definition of the hypergeometric on MATHEMATICA, while the collocation points are sampled from an uniform distribution depending on the range of interest.
The loss function implemented for the model has three components: the loss on the data samples L_{data} is the MSE between the model output and the hypergeometric solution on $\vec{x}_{data}$:

$$L_{data} = \frac{1}{N_{data}} \sum_{x \in \vec{x_{data}}} |y_{nn}(x) - y_{data}|^2 \tag{5.12}$$

The loss on the derivative samples L_{der} is similar to L_{data}, however the predictions are not directly computed by NN, but the model first produces an output on the derivative points ($y_{nn}(x_{der})$), then their derivative are performed by Automatic

Differentiation as explained in Sec. 3.5:

$$L_{der} = \frac{1}{N_{der}} \sum_{x \in \vec{x_{der}}} \left| y'_{nn}(x) - y_{der} \right|^2 \tag{5.13}$$

Finally the third contribution to the total loss is what makes the model "physics informed" by imposing the soft constraint defined by Eq (5.1); given a vector of collocation points $\vec{x}_{coll}$ we can compute for each of them the residuals of the hypergeometric differential equation

$$L_{DE} = \frac{1}{N_{coll}} \sum_{x \in \vec{x}_{coll}} |\Xi(x) - 0|^2 \tag{5.14}$$

all these terms are finally grouped together by a simple linear scalarisation:

$$L_{PINN} = \lambda_{data} L_{data} + \lambda_{der} L_{der} + \lambda_{DE} L_{DE} \tag{5.15}$$

the values of the scalar multiplier will be set dynamically using the ReLoBRaLoss implementation explained as in Sec. 4.3. Finally, for the following example we chose Adam as the model optimizer (which is described in Sec. 3.6) and the learning rate has an initial value of 1×10^{-2} and scheduled to decreased each time the training loss shows no improvements for 10 consecutive epochs.

5.2.1 PINN testing

The training loss is a deceitful metric to test the performance of the Neural Network, as seen in the study of the harmonic oscillator in Sec. 3.8.1: since the training loss is what's used to update the model an independent check is needed to test its performance. We therefore choose to perform a *validation check*: we sample a new set of data $\{\vec{x}_{val}, \vec{y}_{val}\}$ that the model hasn't seen before and at each iteration we compute the MSE w.r.t. the model prediction:

$$L_{val} = \frac{1}{N_{val}} \sum_{x \in \vec{x}_{val}} |y_{nn}(x) - y_{val}|^2 \tag{5.16}$$

we can therefore verify if any increase in accuracy over the training data set actually yields an increase in accuracy over the validation set: if there are no

improvements the NN is overfitting and the training phase can be stopped. For the following examples we'll use $N_{val} = 200$ validation points uniformly distributed in the domain of the input variable . Additional tests are performed after training: a testing data set is created:

$$\{x^i_{test}, y^i_{test}\}, \quad i = 1, ..., N_{test} \tag{5.17}$$

with $N_{test} = 10^3$ points uniformly distributed in the function domain, once again avoiding any overlap with the collocation points, we then use this data set to evaluate the predictive accuracy of the model, for example computing the absolute error between $y_{nn}(x_{test})$ and y_{test}:

$$E_{abs}(y_{nn}(x_{test}), y_{test}) = |y_{nn}(x_{test}) - y_{test}| \tag{5.18}$$

the result is an array of N_{test} values which we'll study both as a distribution and as a function on the input x_{test}.
A similar analysis can be done for the relative error:

$$E_{rel}(y_{nn}(x_{test}), y_{test}) = \left| \frac{y_{nn}(x_{test}) - y_{test}}{y_{test}} \right| \tag{5.19}$$

this is an useful metric to compare the performance for different functions, however, due to the division operation we have to eliminate all the points where y_{test} is vanishing.

5.3 Limiting cases of Hypergeometric function

Now, we apply the above described PINN model set up to the hypergeometric differential equation, whose solution is the hypergeometric function. The hypergeometric function as defined in Eq. (5.2), has three parameters, namely a, b, and c. Here we choose specific values of the parameters a, b and c to study a few limiting cases of the hypergeometric function, namely, i) a polynomial, ii) a rational function with logarithmic components and iii) the complete elliptic integral K.

5.3.1 Polynomial function

We choose the following values of these parameters:

$$a = -2, \quad b = 1, \quad c = 4\,, \tag{5.20}$$

which renders the hypergeometric function to the following polynomial

$${}_2F_1(-2, 1, 4, x) = \frac{1}{10}(10 - 5x + x^2). \tag{5.21}$$

The domain of the variable x together with the range and number of data, validation and test points are:

$$\begin{aligned}
&x \in [0, 20], \\
&x_{data} = 0, \quad N_{data} = 1 \\
&x_{der} \in [0, 5], \quad N_{der} = 9 \\
&x_{coll} \in [0, 20], \quad N_{coll} = 8192 \\
&x_{val} \in [0, 20], \quad N_{val} = 200 \\
&x_{test} \in [0, 20], \quad N_{test} = 10^3
\end{aligned} \tag{5.22}$$

The analytic expression provided in Eq.(5.21) is used to calculate the values of the function and its derivative for $\vec{x}_{data}$, $\vec{x}_{der}$, $\vec{x}_{val}$ and $\vec{x}_{test}$ while the collocation points are generated with an uniform distribution. We apply the PINN algorithm as discussed above with a network architecture having 4 hidden layers with 64 neurons per layer. We choose the activation function as the Gaussian Error Linear Units (GELU). The evolution of the training loss functions with the epochs, as shown in Fig. 5.1 (right panel), shows the convergence toward the predicted solution, as the training loss reaches very small values $\mathcal{O}(10^{-8})$ at the end. The training is stopped after 10^3 epochs. The validation loss also reaches very small values $\mathcal{O}(10^{-9})$ at the end. The solution of the DE obtained by using the PINN method, at the end of the training, are shown in Fig. 5.1 (right panel) along with the distribution of the data points used and the exact analytical solution from Eq.(5.21). We compute the absolute error as defined in Eq.(5.18) for all the values of x in the test dataset and show in the right panel of Fig. 5.2. The mean

value of the absolute error is 9.3×10^{-5}. We also show the distribution of the absolute error in Fig. 5.2 (left panel), which covers a range of $10^{-7} - 10^{-4}$. In this way, the trained network allows us to predict the solution quasi-instantaneously at any point inside the domain of definition of x.

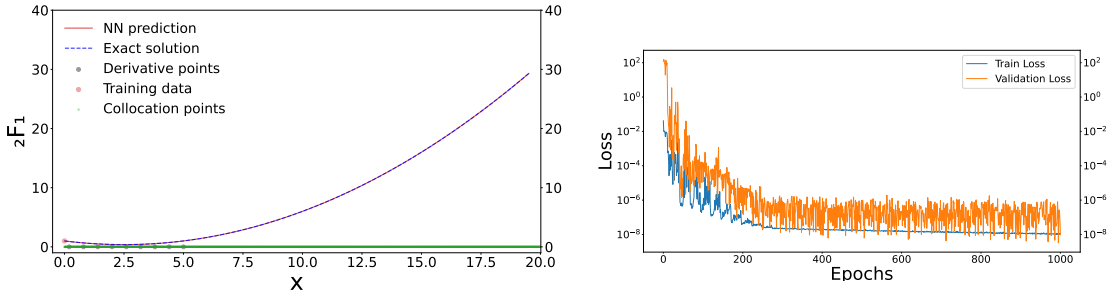


Figure 5.1: (Left panel) PINN prediction with hypergeometric solution and (Right panel) values of the train loss (L_{train}) and validation loss (L_{val}) for different epochs

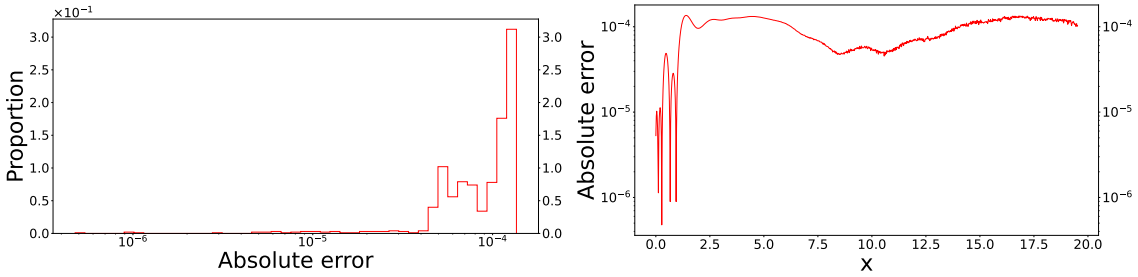


Figure 5.2: (Left panel) the absolute error distribution and (Right panel) the absolute error over the range of values of $\mathbf{x}$

5.3.2 Rational function with logarithmic component

The choice of hypergeometric parameters is:

$$a = 2, \quad b = 5, \quad c = 11 \tag{5.23}$$

the resulting hypergeometric can be expressed in the form:

$$
\begin{aligned}
{}_2F_1(2,5,11,x) = &\frac{3}{2x^{10}}(x(-7560+29820x-44520x^2+30310x^3-8512x^4+ \\
&420x^5+40x^6+5x^7)+840(-1+x)^4(-9+4x)\log(1-x))
\end{aligned}
\tag{5.24}
$$

while it's still a rational function the term with the logarithm makes this case a step beyond the polynomial.
The number and range of both collocation points and data points is the same as before, the only difference is in NN architecture where we choose the hyperbolic tangent as the activation function; the training run lasts 500 epochs and its results are shown in Fig. 5.3.
This time the the training loss reaches an order of $\mathcal{O}(10^{-5})$ and the minimum value of the validation loss is of order $\mathcal{O}(10^{-6})$, however, you may notice how after a certain point in the training the validation loss gets worse even though L_{train} still have a decreasing trend; this encourages another procedure during training to identify the best performing configuration of our model: we keep track of a variable $\hat{L}_{val}$ which store the minimum value of the validation loss, at each epoch t we then compares the new validation loss with it, and if $L_{val}(t) < \hat{L}_{val}$:

- we store the model parameters, including its weights, in a dictionary (a collection of name-value pairs) which will be saved at the end of training

- we update the minimum validation loss value ($\hat{L}_{val} = L_{val}(t)$)

this way after training we can load the dictionary, set the NN weights with their saved values and retrieve the model state at the best performance for further testing.
Using then the best NN performance the distribution of the absolute error is shown in Fig. 5.4 (left panel): we have more dispersion than the previous case, with a mean value of 1×10^{-4}.

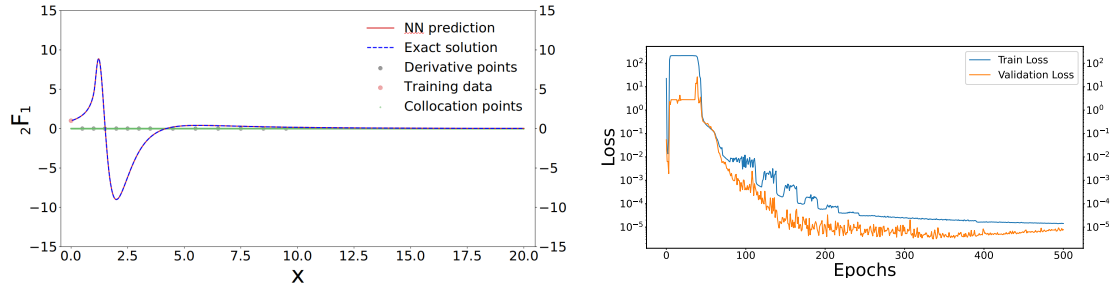


Figure 5.3: (Left panel) PINN prediction with hypergeometric solution and (Right panel) values of the train loss (L_{train}) and validation loss (L_{val}) for different epochs

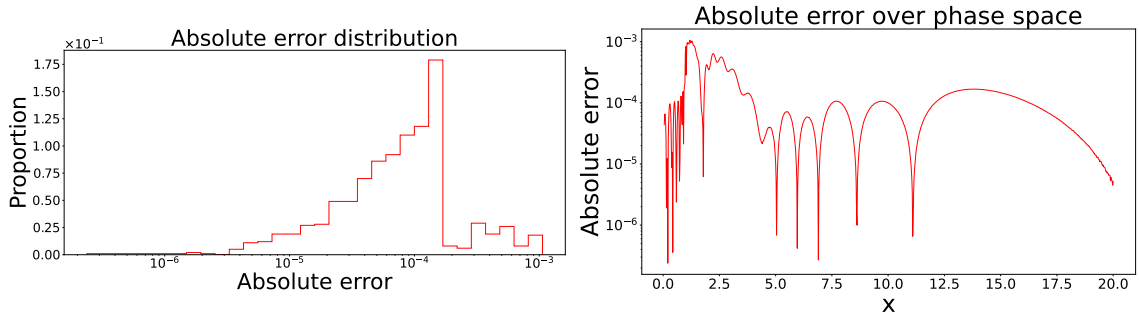


Figure 5.4: (Left panel) Plots of the absolute error distribution and (Right panel) the absolute error over the phase space

5.3.3 Complete Elliptic integral of first kind

Since elliptic integrals have an important role in computation of multi-loop Feynman integrals we aim to the solution of the complete elliptic integral of the first kind K(x), which, as seen in Eq. (5.10) can be defined as a special case of the hypergeometric function with parameters:

$$a = \frac{1}{2}, \quad b = \frac{1}{2}, \quad c = 1 \tag{5.25}$$

$$_2F_1(1/2, 1/2, 1; x) = \frac{2}{\pi} K(\sqrt{x}) \tag{5.26}$$

The elliptic integral diverges for $x \to 1$, this dictates more care in the choice of phase space and data sampling, the domain for the input variable x, the range

and number of data points $\vec{x}_{data}$, derivative points $\vec{x}_{der}$ and collocation points $\vec{x}_{coll}$ are as follow:

$$\begin{aligned} \vec{x} &\in [1.05, 20], \\ x_{data} = 2.1, &\quad N_{data} = 1 \\ \vec{x}_{der} &\in [1.05, 5], \quad N_{der} = 10 \\ \vec{x}_{coll} &\in [1.1, 20], \quad N_{coll} = 8192 \\ \vec{x}_{val} &\in [1.05, 20], \quad N_{val} = 200 \\ \vec{x}_{test} &\in [1.05, 20], \quad N_{test} = 10^3 \end{aligned} \tag{5.27}$$

The results of training are shown in the plots at the end of this section.
The validation loss shown in Fig 5.5 (right panel) shows a decreasing trend up to an order of 10^{-6} except for a sharp increase in the initial epochs due to the instability in the optimization step, the ReLoBraLoss is a significant contribution to ensure a satisfying fit.
The plot of the absolute error in Fig. 5.6 (left panel) shows its distribution peaked on its mean value $E_{abs}^{mean} = 1 \times 10^{-4}$, we observe that for some test points the error gets worse (of order 10^{-2}), their proportion is insignificant but it's worth noting that they are found for $x \to 1$ as shown in Fig. 5.6 (right panel); this is not surprising since for the same limit the hypergeometric diverges and its slope changes rapidly.

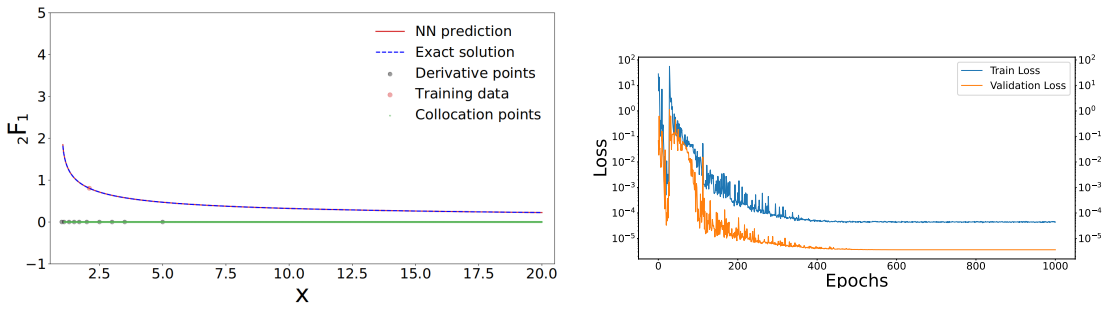


Figure 5.5: (Left panel) PINN prediction with hypergeometric solution and (Right panel) values of the train loss (L_{train}) and validation loss (L_{val}) for different epochs

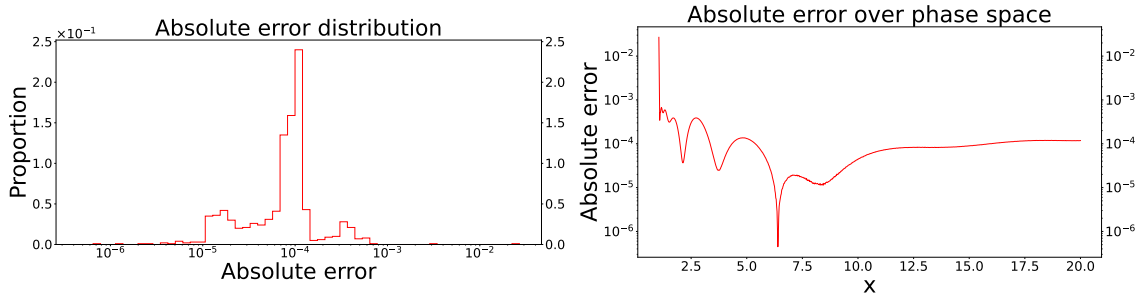


Figure 5.6: Plots of the absolute error distribution (a) and the absolute error over the phase space (b)

5.4 Hypergeometric DE as a system of first order DE

The hypergeometric function demonstrated to be a practical playground for testing the capabilities of a PINN for the solution of a differential equation, however, before directly addressing the evaluation of a Feynman diagram, we can make an intermediate step closer toward this problem which we'll face in chapter 6 onwards; the crux of the matter will be a *system of first order differential equations* (specifically the one satisfied by the Master Integrals w.r.t a kinematic variable), so it will be instructive to consider a simple example of such a problem.
Luckily, the hypergeometric ${}_2F_1$ can still be useful: together with the β function, we can define:

$$\begin{aligned} M_1(x;a,b,c) &= {}_2F_1(a,b-1,c-2,x)\,\beta(b-1,c-b-1) \\ M_2(x;a,b,c) &= {}_2F_1(a+1,b,c-1,x)\,\beta(b,c-b-1)\,(x-1) \end{aligned} \tag{5.28}$$

and these new functions solve the differential equations system:

$$\frac{d}{dx}\begin{pmatrix} M_1 \\ M_2 \end{pmatrix} = \mathbb{A}\begin{pmatrix} M_1 \\ M_2 \end{pmatrix} \tag{5.29}$$

where:

$$\mathbb{A} = \begin{pmatrix} 0, & \frac{a}{x-1} \\ \frac{1-b}{x}, & \frac{-2+c-(-1+a+b)\,x}{(x-1)\,x} \end{pmatrix} \tag{5.30}$$

contains the coefficients of the system in matrix form, then fixing the hypergeometric function parameters $\{a,b,c\}$ we obtain a system of coupled first order

differential equations.
The difference in the code and the analysis w.r.t. the hypergeometric case is relatively small: first the model now outputs two scalars $\{y_{nn,i}\}_{i=1,2}$ representing the prediction of M_1 and M_2, second the data set of the derivative of the functions is not needed since we're dealing with differential equations of the first order; the only data set used $\{\vec{x}_{data}, \vec{y}_{data}\}$ is once again computed knowing the analytic form in Eq.(5.28).
The loss has now only two components: the data loss L_{data} is the same as in Eq.(5.12) while the physics-informed loss is now derived by Eq. (5.30), i.e. given a set of collocation points $\vec{x}_{coll}$:

$$L_{DE} = \frac{1}{2\,N_{coll}} \sum_{x \in \vec{x}_{coll}} \sum_{i=1}^{2} \left| \frac{dy_{nn,i}(x)}{dx} - \mathbb{A}_{ij}\, y_{nn,i}(x) \right|^2 \tag{5.31}$$

Finally a bit of care is required for the choice of some parameters, in particular for $\{a, b, c\}$ and the range of collocation points, which has to be made according to the function domain and the matrix spurious singularities.
Starting from the β function, we know that $\beta(z_1, z_2)$ is defined for $Re(z_1) > 0, Re(z_2) > 0$, but it also has an analytic continuation in the negative values as long as they are not integers so considering the definition in Eq.(5.28), we have:

$$\begin{cases} b - 1 \neq -n \\ b \neq -n \quad n \in N_0 \\ c - b - 1 \neq -n \end{cases} \tag{5.32}$$

which translates into the requirements $b, c \neq -n$ and $b \neq 1$, in our work we consider $\{a = \frac{1}{3}, b = \frac{1}{5}, c = \frac{1}{11}\}$ and $\{a = \frac{1}{2}, b = \frac{1}{2}, c = 4\}$; finally, we observe that the differential equations matrix in Eq. (5.30) has spurious singularities in $x = 0$ and $x = 1$, these are values where its denominators vanish, but they're not necessarily singularities of the solutions, however they should be avoided to not encounter divergences in the loss computation, therefore we'll chose a range for collocation points in the domain $x > 1$.
A validation step will once again be performed dynamically during training and further testing afterwards using the most meaningful metrics for each case.

Case 1 The parameters chosen for the hypergeometric functions in M_1, M_2 are:

$$a = \frac{1}{2}, \quad b = \frac{1}{2}, \quad c = 4 \tag{5.33}$$

The domain data sets are described below:

$$\begin{aligned} \vec{x} &\in [1.05, 50], \\ \vec{x}_{data} &\in [1.05, 5], \quad N_{data} = 10 \\ \vec{x}_{coll} &\in [1.5, 50], \quad N_{coll} = 32768 \\ \vec{x}_{val} &\in [1.05, 50], \quad N_{val} = 500 \\ \vec{x}_{test} &\in [1.05, 50], \quad N_{test} = 10^3 \end{aligned} \tag{5.34}$$

The collocation, validation and test points are all sampled from an uniform distribution, taking care to avoid overleap between them. The NN architecture consists of 4 hidden layers of 32 neurons each, using GELU as an activation function, finally we use the ReLoBRaLoss implementation to set the loss scalar multipliers. The plots for the two functions and the loss are shown in Fig. 5.8 and Fig. 5.7: the mean value of the validation loss reaches an order 10^{-8} with a steady decrease on 200 epochs.
From the absolute errors distribution shown in Fig. 5.9 (left panel) we infer that the model perform equally well for M_1 and M_2, with a mean value of $E_{abs}^{mean} = 2 \times 10^{-4}$, moreover both of the errors drops at the data points and slightly increase with x.

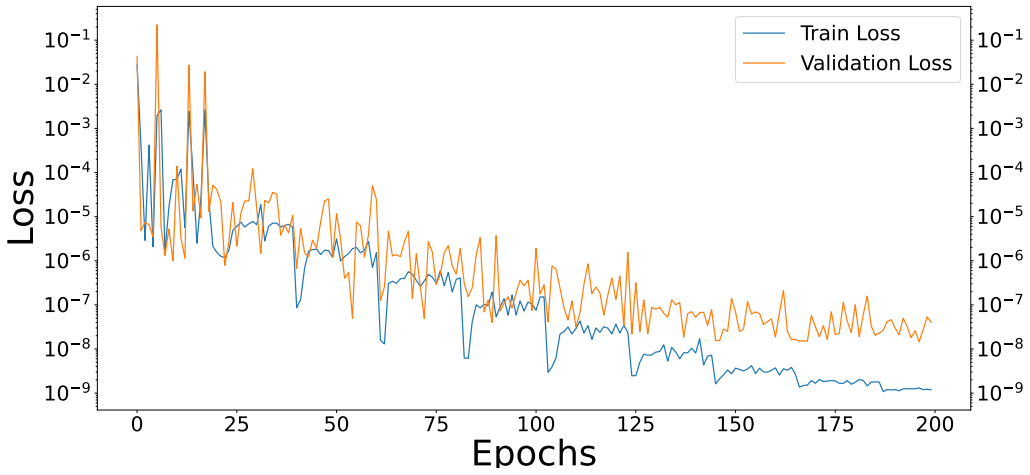


Figure 5.7: Values of the train loss (L_{train}) and validation loss (L_{val}) for different epochs

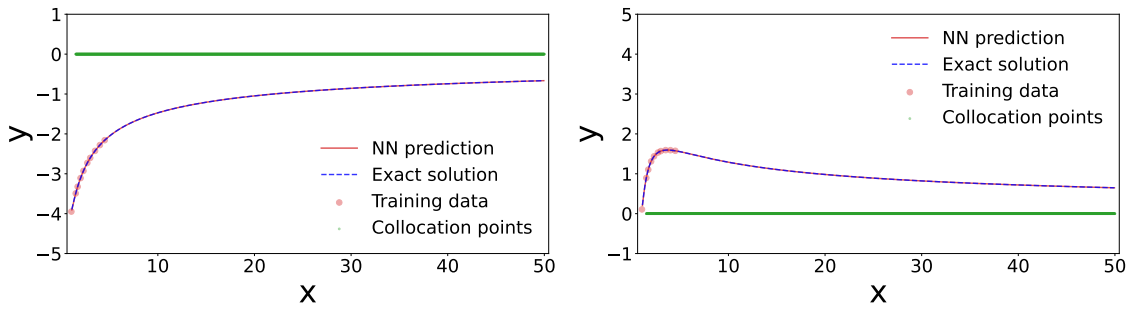


Figure 5.8: PINN prediction and its solution for the function $y = M_1$ (Left panel) and $y = M_2$ (Right panel)

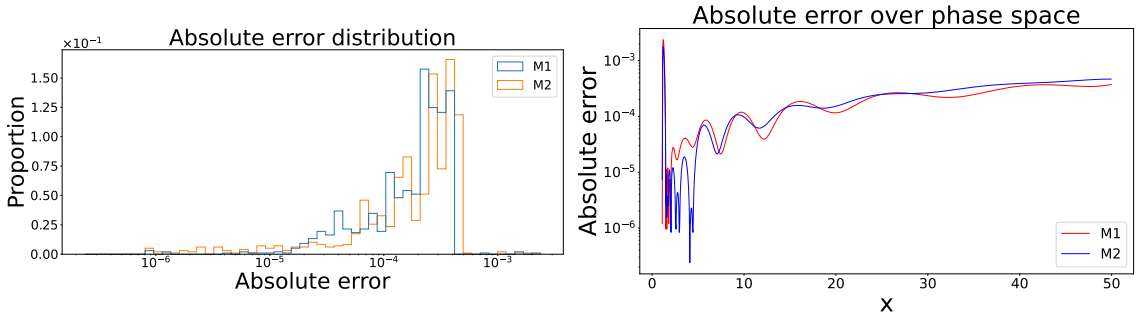


Figure 5.9: (Left panel) absolute error distribution for M_1 and M_2 and (Right panel) absolute error values over phase space

Case 2 The hypergeometric parameters in this example are:

$$a = \frac{1}{3}, \quad b = \frac{1}{5}, \quad c = \frac{1}{11} \tag{5.35}$$

The NN architecture and and data points distribution is the same as the previous example, with the exception of the collocation points:

$$\begin{aligned} \vec{x} &\in [1.05, 50], \\ \vec{x}_{data} &\in [1.05, 5], \quad N_{data} = 10 \\ \vec{x}_{coll} &\in [1.5, 50], \quad N_{coll} = 16364 \\ \vec{x}_{val} &\in [1.05, 50], \quad N_{val} = 500 \\ \vec{x}_{test} &\in [1.05, 50], \quad N_{test} = 10^3 \end{aligned} \tag{5.36}$$

The plots for the two functions and the loss are shown in Fig. 5.11 and Fig. 5.10: while the prediction of the PINN approximate the behaviour of the target functions, the validation loss computed during training is now four order of magnitude worse than the previous example, this is due to the behaviour of the second function, as shown in Fig. 5.11 (right panel), which diverges as its inputs approaches one, so that little variations in the slope of our prediction can produce a significant difference.
This is further corroborated by the absolute and relative errors distribution (Fig. 5.12): while the absolute error for M_2 is clearly worse, the relative errors are actually of the same order, the mean value of these errors are summarized in the table below:

	M_1	M_2
abs	8×10^{-3}	2×10^{-2}
rel	8×10^{-4}	5×10^{-4}

Table 5.1: Mean value of the absolute and relative error for the functions M_1 and M_2.

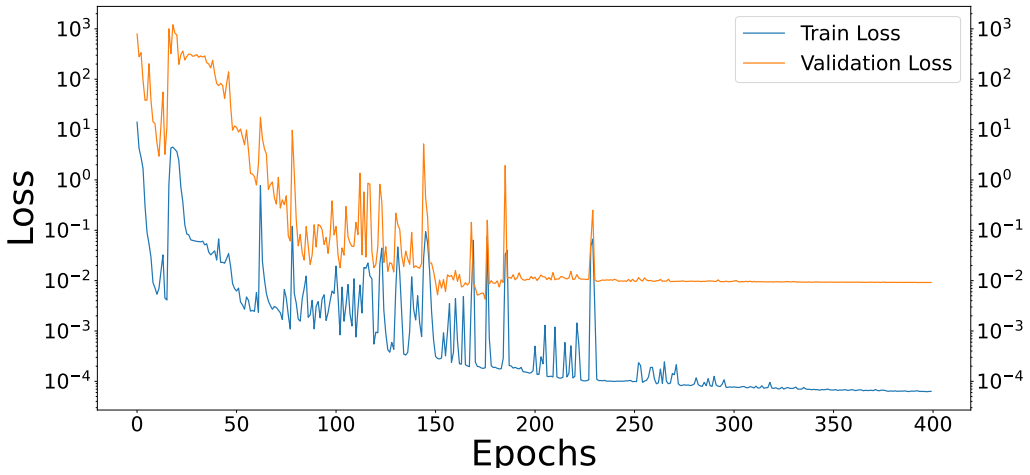


Figure 5.10: Values of the train loss (L_{train}) and validation loss (L_{val}) for different epochs

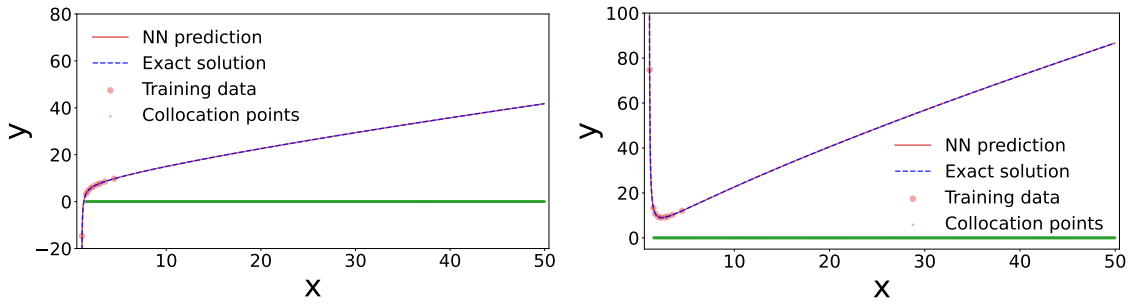


Figure 5.11: PINN prediction and its solution for the function $y = M_1$ (Left panel) and $y = M_2$ (Right panel)

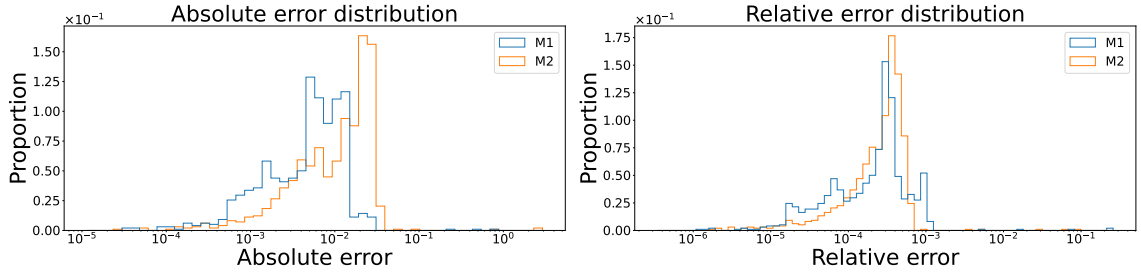


Figure 5.12: (Left panel) absolute error distribution for M_1 and M_2 and (Right panel) absolute error values over phase space

6

Feynman Integrals and PINN

The results of the previous chapter prove a Physics Informed Neural Network to be a promising tool in solving systems described by a Partial Differential Equation, while in chapter 2 we explained how the problem of evaluating Feynman Integrals can be translated into a system of differential equations for few, selected Master Integrals seemingly bringing it within a PINN scope of applicability; for the rest of this Thesis our aim is then to test this remark implementing and training a Physics Informed Neural Network for different Feynman Integrals.
In this chapter we'll focus on the solution for One Loop Feynman Integrals: in section 6.1 we describe the PINN setup for the fit of the Master Integrals differential equation system, and we show the training result of the Massless Box diagram in Sec. 6.2.1 and the Box diagram with one external mass in Sec. 6.2.2

6.1 PINN model for Feynman Integrals

We recall that for a given set of Master Integrals $\vec{M}$, a system of differential equations can be written w.r.t. a mass scale or a kinematical invariant:

$$\frac{\partial \vec{M}(\vec{s},\varepsilon)}{\partial s_i} = \mathbb{A}_{s_i}(\vec{s},\varepsilon)\cdot\vec{M}(\vec{s},\varepsilon) \tag{6.1}$$

where $\mathbb{A}_{s_i}$ is the Connection Matrix, since this is the Physics information we want to introduce in the Neural Network we need to ask which requirements it has to satisfy to make the implementation viable.
First of all note that Feynman (and therefore Master) Integrals typically have complex values, therefore for their computational evaluation we wish to treat the evolution of the real and imaginary parts separately, that is:

$$\begin{aligned} \frac{\partial\, Re\{\vec{M}\}}{\partial s_i} &= \mathbb{A}_{s_i} \cdot Re\{\vec{M}\} \\ \frac{\partial\, Im\{\vec{M}\}}{\partial s_i} &= \mathbb{A}_{s_i} \cdot Im\{\vec{M}\} \end{aligned} \tag{6.2}$$

however this decomposition is possible only if the Connection Matrix has real elements, this can be achieved with a careful choice of the phase space or it can be ensured expressing $\mathbb{A}_{s_i}$ only with rational functions of $\vec{s}$ and ε, which is possible by choosing Master Integrals defined by rational functions themselves (the derivative of a rational function is still rational, so if the l.h.s. of Eq. (6.1) is rational and likewise $\vec{M}$ on the r.h.s. the only acceptable form for $\mathbb{A}_{s_i}$ is to be rational).
Secondly, note the system of differential equations is calculated w.r.t. a kinematic invariant, but both the Master Integrals and the Connection Matrix also depends on the regulator ε, to factorize this dependence we decide to solve the system not for the full expression of Master Integrals but for their Laurent expansion:

$$\vec{M}(\vec{s},\varepsilon) = \sum_{l=-2L} \varepsilon^l \vec{M}^{(l)}(\vec{s}) = \varepsilon^{-\hat{l}} \sum_{l=0}^{\hat{l}} \varepsilon^l \vec{M}^{(l)}(\vec{s}) \tag{6.3}$$

where $\hat{l} = 2L$ which typically corresponds to the lowest power for an expansion of a L-loop Feynman Integral; as a consequence, we require that also the Connection Matrix can be expanded in ε:

$$\mathbb{A}_{s_i}(\vec{s},\varepsilon) = \sum_{j=0}^{j_{\mathbb{A}}} \varepsilon^j \mathbb{A}_{s_i}^{(j)}(\vec{s}) \tag{6.4}$$

with $j_{\mathbb{A}} > 0$, note that this condition while sometimes it's not trivial to achieve is less restrictive than requiring $\mathbb{A}_{s_i}$ to be in canonical form which for the training

of a PINN does not necessarily bring an advantage (roughly speaking the more relations we implement in the Neural Network the more information it has to fit the MIs).
As final result, the differential equation we will implement in the Neural Network will be generally of the form:

$$\frac{\partial \vec{M}^{(l)}(\vec{s})}{\partial s_i} = \sum_{j=0}^{min(j_{\mathbb{A}},l)} \mathbb{A}_{s_i}^{(j)}(\vec{s}) \cdot \vec{M}^{(l-j)}(\vec{s}) \tag{6.5}$$

however we underlined that nothing excludes the presence of an inhomogeneous term $N^{(l)}(\vec{s})$ which would be defined by the solution of the subtopologies for the given integral families, once again expanded in terms of ε, this suggests that a bottom up approach to the solution of the MIs system, that is first solving it for the subtopologies and then addressing the top Master Integrals, is a viable option and it's further explained in 6.1.3.

6.1.1 Phase Space and Dataset

Once the differential equations are set, we have to establish where we want to solve them that is the range of the kinematic invariants where we train our PINN; this choice requires a certain care for two main reasons: the first is that both the equations and their solutions have singularities, these may be physical (where the solutions are actually singular) or spurious, where the Connection Matrix is singular while the MIs stay finite, usually due to the vanishing denominators; regardless of their nature these are points to avoid in the NN training to not encounter divergences.
Secondly, Feynman Integrals are multivalued functions, therefore one has to choose a specific branch for their evaluation and define how to perform an analytical continuation into another kinematic region, we usually avoid this issue by choosing to fit the Master Integrals in only one kinematic region.
In practice for each Feynman Diagram we'll have both pre-generated data, data generated during training at each iteration and data for a post-training analysis of the model performance: the first consists of the boundary points Ω_b and

validation points Ω_{val}:

$$\begin{aligned}\Omega_b &= \{\vec{x}_b^{\,i}, \vec{M}^{(l)}(\vec{x}_b^{\,i})\}, \quad i = 1, ..., N_{data} \\ \Omega_{val} &= \{\vec{x}_{val}^{\,j}, \vec{M}^{(l)}(\vec{x}_{val}^{\,j})\}, \quad j = 1, ..., N_{val}\end{aligned} \tag{6.6}$$

with $l = 0, ..., 2L$, the boundary dataset will be used in the training loss L_{train} therefore contributing to backpropagation, while Ω_{val} will be used in a validation step to independently check the performance of the model in the learning phase. For most of the following examples the datasets are produced using the AMFlow package in MATHEMATICA, when the Master integrals are expressed in terms of multiple polylogarithms (MPLs) their evaluation is performed with HandyG, a Fortran library for fast numerical evaluation of MPLs; the data for the non-planar three points diagram in Sec. 6.3.3 is evaluated directly in MATHEMATICA knowing the analytical solution from [42], while for the Sunrise diagram in Sec. 6.3.2 we used the Python library PySecDec [[43], [44]].
The dynamically computed datasets is instead made of pairs:

$$\Omega_{coll} = \{\vec{x}_{i,coll}, \mathbb{A}^{(j)}_{x_k}(\vec{x}_{i,coll})\} \tag{6.7}$$

with $\mathbb{A}^{(j)}_{x_k}(\vec{x}_{i,coll})$ the components of the Connection Matrix for the kinetic variable x_k, at every order in ε ($j = 0, ..., j_{\mathbb{A}}$) evaluated at the point $\vec{x}_{i,coll}$ ($i = 1, ..., N_{coll}$).
The code implementation of this dataset sampling is an original contribution for this thesis and aimed to grant flexibility and future implementations: at each iteration the values of $\vec{x}_{coll}$ are randomly sampled from an array of so called *base points*, $\vec{x}_{base}$ generated in the phase space region of interest at the beginning of training, with $N_{base} > N_{coll}$; the random sampling is performed by first associating to the base points a probability array p_{base} generated using the Dirichlet distribution which is a generalization of the Beta distribution whose density function is:

$$f(x_1, ..., x_k; \alpha_1, ..., \alpha_k) = \frac{1}{B(\vec{\alpha})} \cdot \prod_{i=1}^{k} x_i^{\alpha_i - 1} \tag{6.8}$$

where $\vec{\alpha}$ is a collection of positive real values, $\vec{x}$ are non negative reals such that $\sum_i x_i = 1$ and $B(\vec{\alpha})$ is the multivariate beta function:

$$B(\vec{\alpha}) = B(\alpha_1, \ldots, \alpha_n) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)\ldots\Gamma(\alpha_n)}{\Gamma(\alpha_1 + \alpha_2 + \ldots + \alpha_n)} \tag{6.9}$$

choosing the value of α_i $(i = 1, ..., N_{base})$ accordingly we can obtain all probabilities values close to $\frac{1}{N_{base}}$ or being wildly different; then the collocation points are simply picked by the base array whose points have a probability of being selected dictated by p_{base}.
Since the base points number has to be greater than the number of collocations points their storage could result in a significant amount of memory usage, however, this set up was chosen because of the possibility to assign different probability weights to different points in phase space. In this way, the model can be guided to focus on regions of phase space that are more critical, moreover it could lead to an easy implementation of a smart-sampling method where the probability array p_{base} is generated according to the model performance at each point (i.e. the loss value at each point) rather than using a distribution as proposed in [45] and [46].
Finally, the testing dataset comprises the values of the Master Integrals ε-components at testing points the model hasn't seen in the training step:

$$\Omega_{test} = \{\vec{x}^j_{test}, \vec{M}^{(l)}(\vec{x}^j_{test})\}, \quad j = 1, ..., N_{test} \tag{6.10}$$

once again, the MIs values are computed using the solution provided by AMFlow.

6.1.2 Training Loss

We're ready to defined the loss function used in our PINN, given the model output $y^{(l)}_{nn;i}$ (i.e. the prediction for the ε^l component of the i-th Master Integrals), we first define the data loss as the MSE w.r.t. the solution at the boundary:

$$L_{data} = \frac{1}{N^*} \sum_{\vec{x}_i \in \Omega_b} \sum_{j=1}^{N_M} \sum_{l=0}^{\hat{l}} \left| y^{(l)}_{nn;j}(\vec{x}_i) - M^{(l)}_j(\vec{x}_i) \right|^2 \quad N^* = N_M \cdot \hat{l} \cdot N_{data} \tag{6.11}$$

where N_M is the number of Master Integrals, the Physics Informed Loss L_{DE} is likewise a MSE function, but computed on Eq. (6.5):

$$L_{DE} = \frac{1}{N^*} \sum_{\vec{x}_i \in \Omega_{coll}} \sum_{j=1}^{N_M} \sum_{k=1}^{n_s} \sum_{l=0}^{\hat{l}} \left| \partial_{x_k} y^{(l)}_{nn;j} - \sum_{r=0}^{min(l, j_{\mathbb{A}})} \sum_{h=1}^{N_M} \mathbb{A}^{(r)}_{x_k,h}(\vec{x}_i) y^{(l-r)}_{nn;h}(\vec{x}_i) \right|^2 \tag{6.12}$$

where n_s is the number of independent kinematic invariants and $N^* = N_{coll} \times N_M \times n_s \times \hat{l}$, the total loss function is then the linear combination of the two previous terms:

$$L_{train} = \lambda_{data} L_{data} + \lambda_{DE} L_{DE} \tag{6.13}$$

setting the values of λ_{data} and λ_{DE} is important whenever the two loss contribution have different scales, which usually happens if the solutions and their derivative are of different order, this is not always the case for the Feynman Integrals considered, however we still use the ReLoBRaLo implementation described in Sec. 4.3.

6.1.3 PINN model with inhomogeneous term

As original contribution of the work presented in this thesis we include the possibility of a inhomogeneous term in the MIs differential equations system.
We previously showed how this system can be written making the presence of the subtopologies Master Integrals $\vec{N}$ explicit:

$$\frac{\partial \vec{M}}{\partial s_\alpha} = \mathbb{A}_\alpha \cdot \vec{M} + \mathbb{B}_\alpha \cdot \vec{N} \tag{6.14}$$

this suggested a bottom up approach: if the subtopologies MIs are solved first, they can be explicitly expressed as functions of the kinematics invariants:

$$\frac{\partial \vec{M}}{\partial s_\alpha} = \mathbb{A}_\alpha \cdot \vec{M} + \vec{f}(\vec{s}) \tag{6.15}$$

This equation is what we actually implement in the code: first of all when the inhomogeneous term is needed we include it in the collocation points sampling :

$$\Omega_{coll} = \{\vec{x}_{i,coll},\ \ \mathbb{A}^{(j)}_{x_k}(\vec{x}_{i,coll}),\ \ \vec{f}_p(\vec{x}_{i,coll})\} \tag{6.16}$$

where p is an index to take into account whether we are fitting for the real or imaginary part of the MIs; then the equation (6.12) for the Physics informed loss becomes:

$$L_{DE} = \frac{1}{N^*} \sum_{\vec{x}_i \in \Omega_{coll}} \sum_{j=1}^{N_M} \sum_{k=1}^{n_s} \sum_{l=0}^{\hat{l}} \left| \partial_{x_k} y^{(l)}_{nn;j} - \sum_{r=0}^{min(l, j_{\mathbb{A}})} \sum_{h=1}^{N_M} \mathbb{A}^{(r)}_{x_k,h}(\vec{x}_i) y^{(l-r)}_{nn;h}(\vec{x}_i) - f^{(l)}_p(\vec{x}_i) \right|^2 \tag{6.17}$$

with $N^* = N_{coll} \times N_M \times n_s \times \hat{l}$ as before.

6.1.4 Analysis of training results

While the validation loss is an useful metric to check the model performance during training, we still wish to test the accuracy of the NN predictions whenever we know the solution: for example once the Neural Network is trained we can compute its output on the test points $\vec{y}_{nn}(\vec{x}_{test})$ and compare it to the target $\vec{y}_{test}$ by computing the absolute and relative difference as described in Eq. (5.18), (5.19), however in this case we deal with multi-dimensional functions:

$$\begin{aligned} \vec{E}_{abs}(\vec{y}_{nn}(\vec{x}_{test}), \vec{y}_{test}) &= |\vec{y}_{nn}(\vec{x}_{test}) - \vec{y}_{test}| \\ \vec{E}_{rel}(\vec{y}_{nn}(\vec{x}_{test}), \vec{y}_{test}) &= \left| \frac{\vec{y}_{nn}(\vec{x}_{test}) - \vec{y}_{test}}{\vec{y}_{test}} \right| \end{aligned} \tag{6.18}$$

the results are now matrices of dimension $(N_{test}, N_\varepsilon \times N_M)$ (i.e. with a value for every ε-component of every Master Integral at each test point), in this way we can study the errors grouped by whichever parameter we're interested in, by ε-order or by Master Integrals, that is:

$$\begin{aligned} E^{(l)}_{abs} &= \frac{1}{N_M} \frac{1}{N_{test}} \sum_{i=1}^{N_M} \sum_{j=1}^{N_{test}} \left| y^{(l)}_{nn;i}(x_j) - y^{(l)}_{test;i}(x_j) \right|, \quad l = l_{min}, ..., l_{max} \\ E^{i}_{abs} &= \frac{1}{N_\varepsilon} \frac{1}{N_{test}} \sum_{l=1}^{N_\varepsilon} \sum_{j=1}^{N_{test}} \left| y^{(l)}_{nn;i}(x_j) - y^{(l)}_{test;i}(x_j) \right|, \quad i = 1, ..., N_M \end{aligned} \tag{6.19}$$

and similarly for the relative error; finally we can check the model performance in different regions of the phase space.
Once again it's worth noting that in the relative error we have to eliminate all

the points where $\vec{y}_{test}$ is vanishing, this could impact some Master Integrals more than others.

6.2 Results for One-Loop Feynman integrals

6.2.1 Massless Box diagram

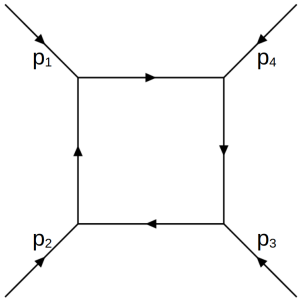


Figure 6.1: Diagram representing the Massless Box Integral family

In this section, we consider the one-loop four-point massless box diagram, where the external momenta p_i's are taken to be all ingoing and massless. The momentums satisfy the following kinematical constraints:

$$\sum_{i=1}^{4} p_i^{\nu} = 0, \qquad p_i^2 = 0\,, \qquad i = 1, \cdots, 4. \tag{6.20}$$

We define the Mandelstam invariants as:

$$s = (p_1 + p_2)^2, \quad t = (p_2 + p_3)^2, \quad u = (p_1 + p_4)^2\,. \tag{6.21}$$

since they satisfy the relation $s + t + u = 0$ we can choose $\vec{s} = \{s_0, s_1\} = \{s, t\}$ as independent variables, and working in the s_0 channel we can use for the PINN model the dimensionless variable $x = c_x \frac{s_1}{s_0}$ (with c_x a constant).
The family of integrals of the Massless Box is defined by:

$$I_{n_1,n_2,n_3,n_4}(s,t,\varepsilon) = \int \frac{d^d k}{i\pi^{\frac{D}{2}}} \frac{\mu^{4-D}}{D_1^{n_1} D_2^{n_2} D_3^{n_3} D_4^{n_4}} \tag{6.22}$$

where μ is a dimensional regularisation scale that we can set to 1 (its dependence can always be recovered by dimensional analysis) and $\vec{n} = \{n_1, n_2, n_3, n_4\}$ is the

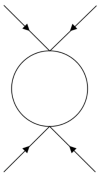
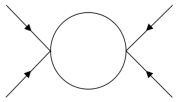
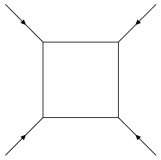
set of integer powers for the inverse propagators D_i, these are defined as:

$$D_1 = (k - p_1)^2, \quad D_2 = k^2, \quad D_3 = (k + p_2)^2, \quad D_4 = (k + p_2 + p_4)^2. \tag{6.23}$$

with k as the loop momentum.
We produce and solve the IBP relations using FiniteFlow+LiteRed, identifying three Master Integrals:

$$\vec{F}(s, t, \varepsilon) = \{I_{0,1,0,1}, \ I_{1,0,1,0}, \ I_{1,1,1,1}\} \ , \tag{6.24}$$

where

$I_{0,1,0,1} =$  , $I_{1,0,1,0} =$  , $I_{1,1,1,1} =$  . (6.25)

However, these Integrals are not dimensionless and by taking advantage of the freedom in choosing a MIs basis, we decide to rescale them multiplying them by a suitable power of t:

$$\vec{M}(s, t, \varepsilon) = \{M_1, M_2, M_3\} = \{t^{\varepsilon} I_{0,1,0,1}, \ t^{\varepsilon} I_{1,0,1,0}, \ t^{(2+\varepsilon)} I_{1,1,1,1}\} \tag{6.26}$$

The differential equation system obeyed by these set of Master integrals is:

$$\frac{d\vec{M}}{dx} = \mathbb{A}_x \cdot \vec{M} \tag{6.27}$$

where $\mathbb{A}_x$ is defined in A.2.
We solve for the MIs truncated Laurent expansion $M_i \sim \sum_{l=l_{min}}^{l_{max}} \varepsilon^l M_i^{(l)}$: the lowest power in the sum is $l_{min} = -\hat{l} = -2L = -2$, while as the upper limit for a one loop diagram we could be satisfied to fit up to the zero-th order in ε (since we don't need higher orders to regulate divergences), however to test the capabilities of the NN model we'll solve up to $l_{max} = \varepsilon^2$, so that we truncate the series at $N_\varepsilon = 2\hat{l} + 1 = 5$.
The connection matrix is linear in ε, and checking its expression in A.2 it has

singularities in:

$$x = 0, \quad x = -1\,. \tag{6.28}$$

The first one corresponds to the physical singularity $s_1 = 0$ and the second to the spurious singularity $s_0 + s_1 = 0$, additionally there's a second spurious singularity for $s_0 = 0$, the latter is avoided by working in the s_0 channel, while the ones in Eq. (6.28) are avoided choosing $x < -1$. The training data consists of two points at the boundary of the phase space while the testing data consists of 10^5 points uniformly distributed.

The full list of hyperparameters for the PINN training is summarized in the table 6.1. The "Neural Network" section describes the PINN architecture which is very simple: its input is the scalar variable x, the output will consists of $N_M \times N_\varepsilon = 15$ values, finally we'll use just three hidden layers of 32 neurons each.

In the "Optimizer" section we define the parameters involved in the backpropagation step: the optimizer algorithm (Adam) was described in Sec. 3.6, while the scheduler ReduceOnPlateu simply reduces the learning rate value when the training loss doesn't decrease for a number of consecutive epochs defined by its *patience*, once the learning rate is reduced the scheduler waits a number of epochs set by its *cooldown* parameter before checking the loss again, and if the learning rate meets a minimum values (*minimum lr*) the scheduler stops.

The "Sampling" section is about the selection of collocation points: before training a set of base points (of number equal to the *Base points size*) is generated, at each epoch the we choose just a subset, or partition, of them as collocation points (so *Partition* = 1) which is fed into the NN in batches whose number of points is defined by *Batch size*, the *Partition size* parameter defines the number of batches so that the total number of collocation points is *Partition Size* $\times$ *Batch Size* = 65536.

Finally we use the ReLoBRaLoss implementation described in Sec. 4.3 to dinamically set the loss scalar multipliers.

Hyperparameter	Value
Neural Network	
input size	1
output size	15
hidden layers	[32, 32, 32]
Activation function	GELU
Optimizer	
Algorithm	Adam
Scheduler	ReduceOnPlateau
Initial lr	$1 \cdot 10^{-2}$
patience	5
cooldown	5
Minimum lr	$1 \cdot 10^{-15}$

Hyperparameter	Value
Sampling	
Base points size	$2 \cdot 10^5$
Samples per partition	65536
Partition size	1024
Batch size	64
Partition	1
Advanced options	
ReLoBRaLoss	True
τ	0.5
α	0.999

Table 6.1: Neural Network architecture for the Massless Box family.

6.2.1.1 Training Results

The training lasted 400 epochs and its statistics are shown in the plots at the end of this section. In Fig. 6.2 is shown the performance of the NN model during training: there is a steady, decreasing trend up to 300 epochs, where the losses stabilize on the values $L_{train} \approx 10^{-7}$ and $L_{val} \approx 10^{-8}$ and no more improvements are possible since the learning rate becomes too small to update the model state significantly, several training sessions with NN weights initialized randomly lead to results of the same orders, so we consider this an exemplary fit.

The absolute errors for the different ε-component in Fig. 6.3 (left panel) shows similar distribution, proving the model to have similar performance at every order

in ε regardless of their complexity, they peak close to their mean value at 9×10^{-5} and in a range between $10^{-8} - 10^{-4}$. A similar study can be done for the relative error, whose distribution is showed in Fig. 6.3 (right panel): here the mean value is of the same order of the absolute error (8.5×10^{-5}), however there is more dispersion, and specifically the ε^{-1} order (j = 1 in the plot) has the worst result.
We also perform the training for the MIs imaginary part: the model hyperparameters and the Physics Informed Loss are the same, but we obviously changed the boundary and testing points.
The plot in Fig. 6.4 compares the absolute error distribution of the real and imaginary part: it's clear that the fit was even more successful for the imaginary part, with a mean absolute error of 4×10^{-5}.
Finally to further compare the NN prediction with the known solution we can take advantage of the fact that the Master Integrals are considered functions of just one variable which allows a clear graphical representation: in Fig. 6.5, 6.6 and 6.7 we plot the first three ε-components of the Master Integrals analytic solution with their respective NN output: as you can see the model successfully reproduce their behaviour and while this is trivial for the t-channel and the first two order of the s-channel, their values span three orders of magnitude yet even a simple PINN architecture was sufficient for the fit.

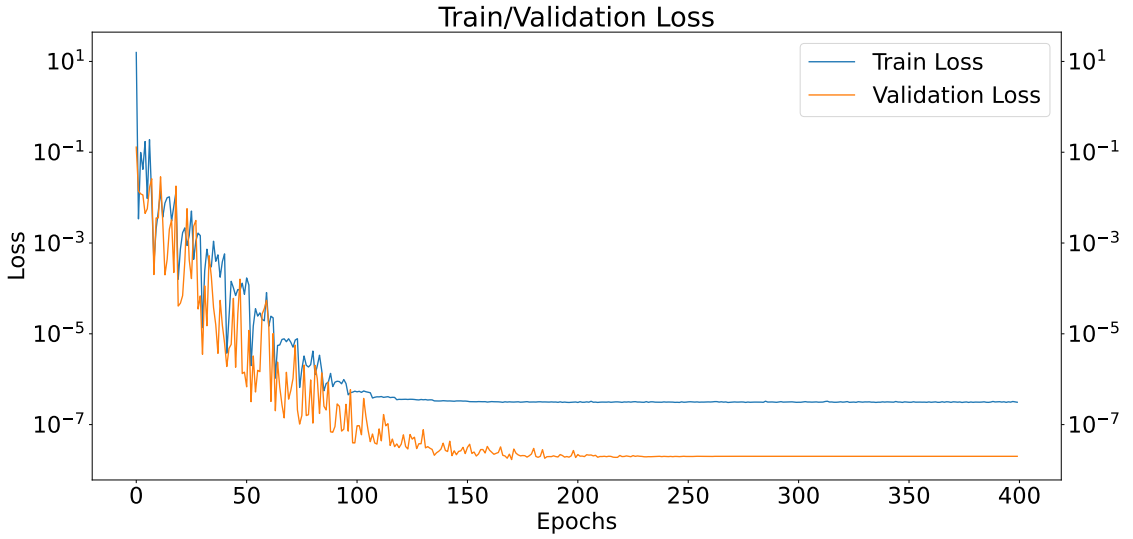


Figure 6.2: Training and test loss over different epochs for the massless Box diagram.

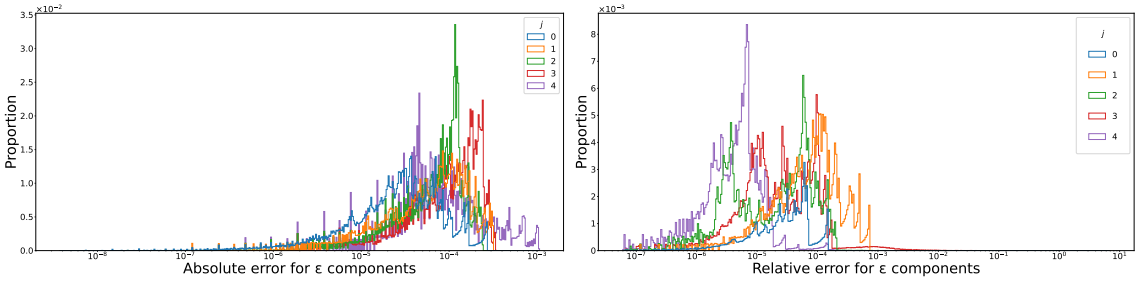


Figure 6.3: (Left panel) Absolute error distribution for different order of Master Integrals and (Right panel) relative error distribution for different order of Master Integrals for the massless Box diagram.

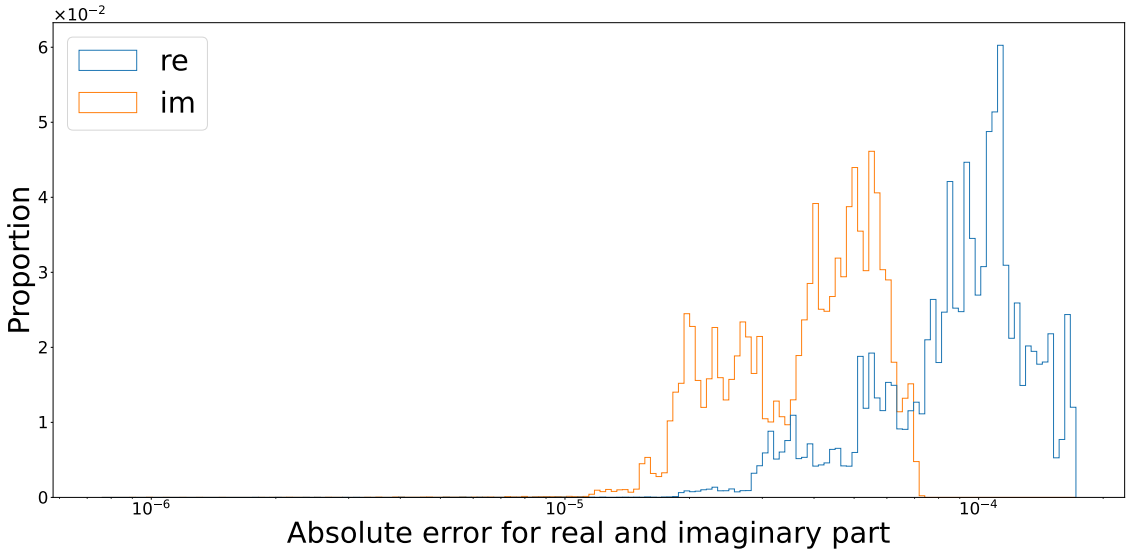


Figure 6.4: Absolute error distribution for the real and imaginary part for the massless Box diagram.

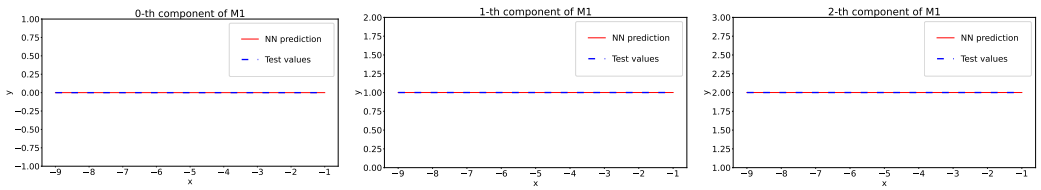


Figure 6.5: Plot of the NN prediction and the actual solution for the first Master Integral

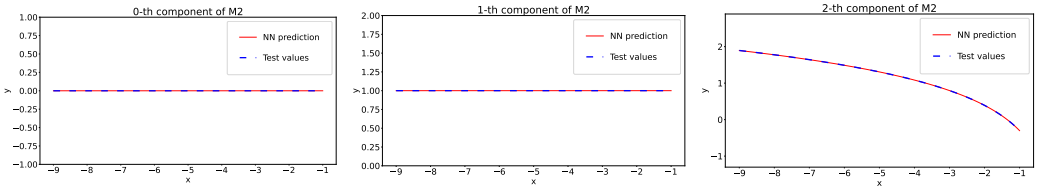


Figure 6.6: Plot of the NN prediction and the actual solution for the second Master Integral

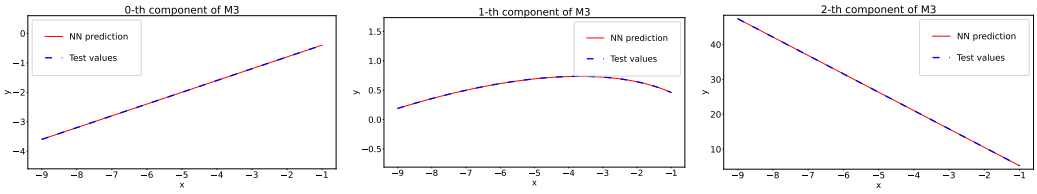


Figure 6.7: Plot of the NN prediction and the actual solution for the third Master Integral

6.2.2 One-mass Box diagram

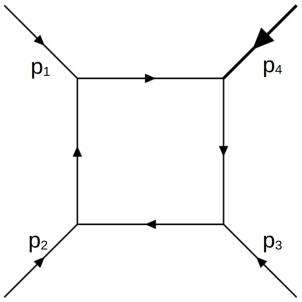


Figure 6.8: Diagram representing the One-mass Box Integral family

A sensible step-up from the previous example is to make one of the external line massive, the momenta are still considered ingoing and satisfy:

$$\begin{aligned} &\sum_{i=1}^{4} p_i = 0, \\ &p_i^2 = 0, \quad i = 1,2,3 \\ &p_4^2 = m^2, \end{aligned} \tag{6.29}$$

the Mandelstam variable are the same, but now their sum is equal to the mass squared:

$$\begin{aligned} &s = (p_1 + p_2)^2, \quad t = (p_2 + p_3)^2, \quad u = (p_1 + p_4)^2, \\ &s + t + u = m^2, \end{aligned} \tag{6.30}$$

the mass scale by itself is a kinematic invariant $s_m = p_4^2 = m^2$, we can select three of them as variable and we choose to use $\vec{s} = \{s_0, s_1, s_2\} = \{s, t, s_m\}$, moreover we choose to work in the s_0 channel so that the PINN model depends only on $\{s_1, s_2\}$. The integrals family of the Box with one external mass is defined by:

$$I_{n_1,n_2,n_3,n_4}(s,t,\varepsilon) = \int \frac{d^d k}{i\pi^{\frac{D}{2}}} \frac{\mu^{4-D}}{D_1^{n_1} D_2^{n_2} D_3^{n_3} D_4^{n_4}} \tag{6.31}$$

with the propagators defined as:

$$D_1 = (k - p_1)^2, \quad D_2 = k^2, \quad D_3 = (k + p_2)^2, \quad D_4 = (k + p_2 + p_4)^2. \tag{6.32}$$

The topology reduction results in identifying four Master Integrals:

$$\vec{F}(\vec{s}, \varepsilon) = \{I_{1,1,1,1};\ I_{1,0,1,0};\ I_{0,1,0,1};\ I_{1,0,0,1}\} \tag{6.33}$$

which will rescale in order to get a dimensionless system:

$$\vec{M}(\vec{s}, \varepsilon) = \{s_1\, s_2^{1+\varepsilon} I_{1,1,1,1};\ s_1\, s_2^{-1+\varepsilon} I_{1,0,1,0};\ s_0^{-1}\, s_2^{1+\varepsilon} I_{0,1,0,1};\ s_2^{\varepsilon} I_{1,0,0,1}\} \tag{6.34}$$

The systems of differential equation for the Master Integrals $\vec{M}$ are:

$$\frac{\partial \vec{M}}{\partial s_1} = \mathbb{A}_{s_1} \cdot \vec{M} \tag{6.35}$$

$$\frac{\partial \vec{M}}{\partial s_2} = \mathbb{A}_{s_2} \cdot \vec{M} \tag{6.36}$$

with $\mathbb{A}_{s_1}$ and $\mathbb{A}_{s_2}$ defined in A.5; we fit for 5 orders in ε, starting from $l_{min} = -\hat{l} = -2L = -2$ up to ε^2 so that the model output have $N_M \times N_\varepsilon = 5 \times 4 = 20$ components.
With the choice of MIs in Eq. (6.34) the Connection Matrices are still linear in ε and their denominators vanish for:

$$\begin{aligned} &s_2 = 0, \quad s_0 = 0, \quad s_1 = 0, \\ &s_2 = s_0 + s_1, \quad s_2 = s_0, \quad s_2 = s_1, \end{aligned} \tag{6.37}$$

where the first is a physical singularity and it's avoided setting a non-zero mass, the second to the fourth are also physical singularities while the last two are just spurious singularities, all of them can be avoided with a suitable choice of phase space:

$$s_0 = const., \quad s_1 < 0, \quad s_2 > 0, \quad s_0 + s_1 - s_2 > 0, \tag{6.38}$$

The boundary dataset consists of just three points and the testing dataset has 10^5 points uniformly distributed.
The full list of hyperparameters for the PINN training is summarized in the table

6.2, since Massive Box is a more complex example than the previous one, the PINN architecture is larger: the neurons of each hidden layers, the batch size of the collocation points and the number of batches are doubled.

Hyperparameter	**Value**
Neural Network	
input size	2
output size	20
hidden layers	[64, 64, 64]
Activation function	GELU
Optimizer	
Algorithm	Adam
Scheduler	ReduceOnPlateau
Initial lr	$1 \cdot 10^{-3}$
factor	0.5
patience	10
cooldown	10
Minimum lr	$1 \cdot 10^{-15}$

Hyperparameter	**Value**
Sampling	
Base points size	$2 \cdot 10^5$
Samples per partition	262144
Partition size	2048
Batch size	128
Partition	1
Advanced options	
ReLoBRaLoss	True
τ	1
α	0.999

Table 6.2: Neural Network architecture for the Massive Box family.

6.2.2.1 Training Results

The training lasted 300 epochs and the resulting statistics are presented in the plots at the end of this section.
In Fig. 6.9 the training and validation loss are shown, they both have decreasing trends up to values of $\mathcal{O}(10^{-7})$ for L_{train} and $\mathcal{O}(10^{-8})$ for L_{val}.

The absolute error plot in Fig. 6.10 (left panel) reveals a similar distribution for the different ε-components, with a mean value of $E_{abs}^{mean} = 1 \times 10^{-4}$ while the relative error is showed in Fig. 6.10 (right panel), this has more dispersion, with a mean value of $E_{rel}^{mean} = 3 \times 10^{-4}$.
Finally, Fig. 6.11 compares the absolute error distribution for the real and imaginary parts: the latter has a wider distribution, however its mean absolute error still equals to 1×10^{-4}.

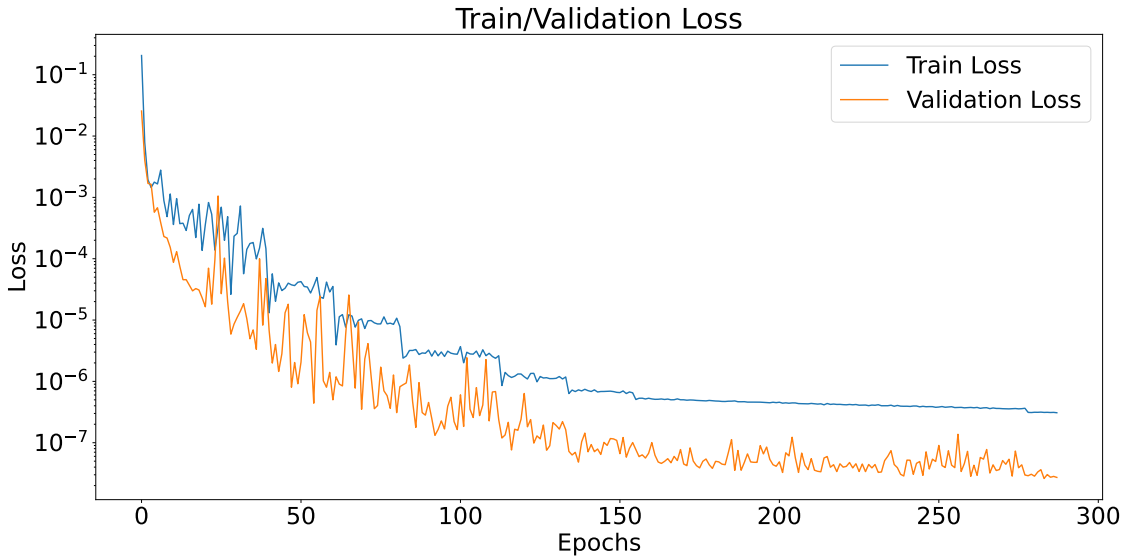


Figure 6.9: Values of the Training and Validation Loss over different epochs for the one-mass Box diagram.

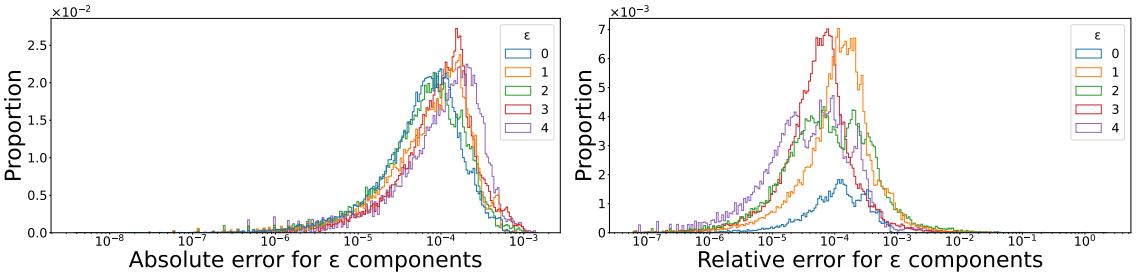


Figure 6.10: (Left panel) Absolute error distribution and (Right panel) relative error distribution for different orders of the Master Integrals of the one-mass Box diagram

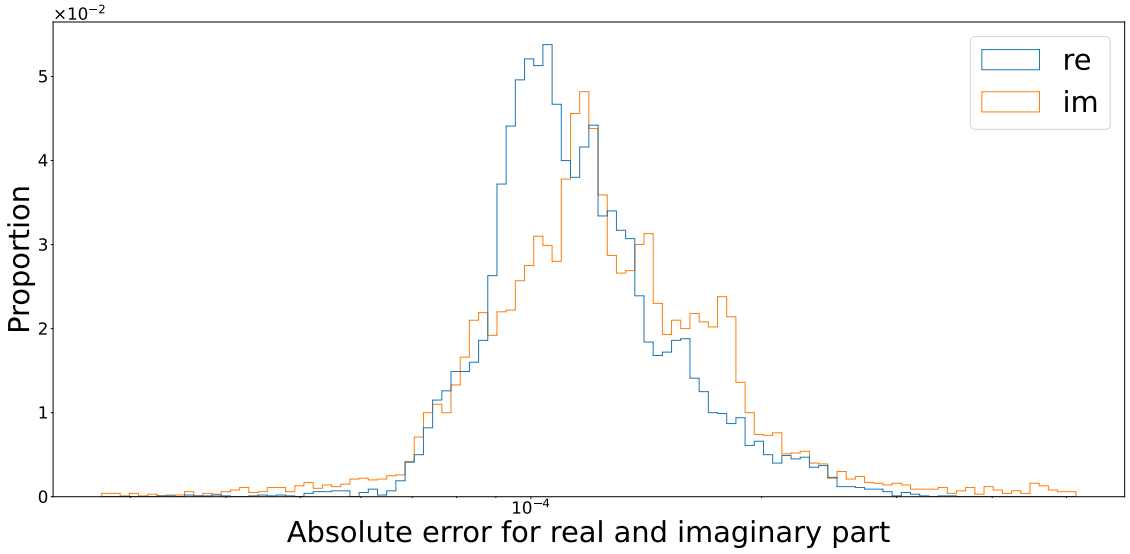


Figure 6.11: Absolute error for real and imaginary part of the one-mass Box diagram MIs.

6.3 Results for Two-Loop Feynman Integrals

This chapter will be focused on the solution of Two Loops Feynman Integrals: in 6.3.1 we show the training results of the One-mass Double Box diagram, an example of a non planar three points diagram is studied in 6.3.3 and the equal-mass Sunrise diagram in 6.3.2.
This section includes an important test for our PINN implementation: indeed all previous Feynman Integrals we've studied can be expressed by multipolylogarithms (MPLs), however the analytic expression of the last two diagrams require more involved function. Both the non planar and the equal-mass Sunrise diagram can be expressed in terms of elliptic integrals: therefore the ability to replicate these integrals would prove the usefulness of the neural network for the study of multi-loop diagrams.

6.3.1 One-mass Double Box diagram

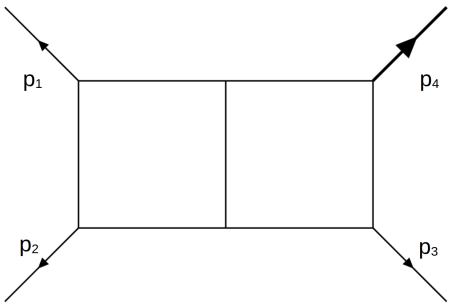


Figure 6.12: Diagram representing the One Mass Double Box Integral family

For the first example of a multi-loop diagram we consider the Double Box with one external mass, the external momenta p_i are taken to be outgoing and satisfying:

$$\begin{aligned} p_i^2 &= 0, \quad i = 1, 2, 3 \\ p_4^2 &= m^2, \end{aligned} \tag{6.39}$$

The kinematic invariants are defined by the Mandelstam variables and the mass scale:

$$s = (p_1 + p_2)^2, \;\; t = (p_2 + p_3)^2, \;\; u = (p_1 + p_4)^2, \;\; s_m = m^2, \tag{6.40}$$

then we can choose any three of them as independent variables, we'll select $\vec{s} = \{s_0, s_1, s_2\} = \{s, t, s_m\}$, moreover we consider the s_0 channel so that the PINN model depends only on $\{s_1, s_2\}$.
The Feynamn Integral associated to the Double Box with one Massive external leg is:

$$I_{\vec{n}}(\vec{s}, \varepsilon) = \int \frac{d^d k_1}{i\pi^{\frac{D}{2}}} \frac{d^d k_2}{i\pi^{\frac{D}{2}}} \frac{\mu^{4-D}}{D_1^{n_1} D_2^{n_2} D_3^{n_3} D_4^{n_4} D_5^{n_5} D_6^{n_6} D_7^{n_7} D_8^{n_8} D_9^{n_9}} \tag{6.41}$$

where the propagators are defined as:

$$\begin{aligned} D_1 = k_1^2, \quad D_2 = (k_1 - p_1)^2, \quad D_3 = (k_1 - p_1 - p_2)^2 \\ D_4 = k_2^2, \quad D_5 = (k_2 - p_4)^2, \quad D_6 = (k_2 + p_1 + p_2)^2 \\ D_7 = (k_1 + k_2)^2, \quad D_8 = (k_1 + p_4)^2, \quad D_9 = (k_2 + p_1)^2, \end{aligned}$$

with $\{k_i\}_{i=1,2}$ as the loop momenta. The solution of the IBP relations with FiniteFlow+LiteRed indentifies $N_M = 18$ Master Integrals:

$$\begin{aligned} \vec{M}(\vec{s}, \varepsilon) = \{ & I_{1,1,1,1,1,1,1,0,-1}, I_{1,1,1,1,1,1,1,0,0}, I_{1,1,1,1,1,1,0,0,0}, \\ & I_{0,1,1,1,1,1,1,0,0}, I_{0,1,1,1,1,1,1,0,-1}, I_{1,1,0,1,1,1,1,0,0}, I_{1,1,1,1,0,1,1,0,0}, I_{1,1,0,1,0,1,1,0,0}, \\ & I_{1,1,1,1,0,1,-1,0,0}, I_{1,1,1,1,1,0,1,0,0}, I_{1,0,1,0,1,1,1,0,0}, I_{1,1,1,1,1,0,1,-1,0}, I_{0,1,1,0,1,1,1,0,0}, \\ & I_{1,0,1,1,1,0,1,0,0}, I_{0,1,1,1,1,0,1,0,0}, I_{1,1,0,1,1,0,1,0,0}, I_{1,0,1,1,0,1,1,0,0}, I_{1,0,0,0,1,0,1,0,0}\} \end{aligned} \tag{6.42}$$

The system of differential equations for $\vec{M}$ are:

$$\frac{\partial \vec{M}}{\partial s_1} = \mathbb{A}_{s_1} \cdot \vec{M} \tag{6.43}$$

$$\frac{\partial \vec{M}}{\partial s_2} = \mathbb{A}_{s_2} \cdot \vec{M} \tag{6.44}$$

where $\mathbb{A}_{s_1}$ is defined in A.8; the MIs Laurent expansions start at $l_{min} = -4$ and end at $l_{max} = 0$ so that $N_\varepsilon = 5$ and the PINN output have $N_M \times N_\varepsilon = 90$ scalar values.
With the choice of MIs above the Connection Matrices are linear in ε, but once again we don't try to put it in a canonical form, looking at its denominators, we

see that they vanish for:

$$\begin{aligned} s_0 - s_2 &= 0 \\ s_1 - s_2 &= 0 \\ s_1 + s_2 &= 0 \\ s_0 + s_1 - s_2 &= 0 \end{aligned} \tag{6.45}$$

the first three are spurious singularities while the last one is a physical one, the other physical singularities are $\{s_i = 0\}_{i=0,1,2}$, to define the phase space we choose to constrain the variables $\{s_1, s_2\}$ to obey the following relations:

$$s_2 > 0, \quad s_1 < 0, \quad s_0 + s_1 - s_2 > 0, \tag{6.46}$$

so that all singularities are outside or at the boundary of the training phase space region.
The boundary dataset consists of 6 points at the edges of the phase space and the testing dataset has 10^5 points uniformly distributed.
The specifics of the PINN training are summarized in table 6.3: due to the large size of the output the NN architecture has now 256 neurons per hidden layers; the collocation points are fed in the model with batches of 256 points with a total number of $N_{coll} = 131072$ points uniformly distributed in the phase space region, we note that the complexity of this case can take advantage of the increased parallelism of a GPUs rig.

Hyperparameter	Value
Neural Network	
input size	2
output size	90
hidden layers	[256, 256, 256]
Activation function	GELU
Optimizer	
Algorithm	Adam
Scheduler	ReduceOnPlateau
Initial lr	$1 \cdot 10^{-2}$
factor	0.5
patience	6
cooldown	0
Minimum lr	$1 \cdot 10^{-15}$

Hyperparameter	Value
Sampling	
Base points size	$4 \cdot 10^5$
Samples per partition	131072
Partition size	512
Batch size	256
Partition	1
Advanced options	
ReLoBRaLoss	True
τ	0.5
α	0.99

Table 6.3: Neural Network architecture for the one-mass Double Box diagram.

6.3.1.1 Training result

The training lasted 300 epochs and its results are shown in the plot at the end of this section.
The training and validation loss shown in Fig. 6.13 follow the same trend with final values of $\mathcal{O}(10^{-4})$ for L_{train} and $\mathcal{O}(10^{-5})$ for L_{val}, the absolute error in Fig. 6.14 (left panel) has a mean value of $E_{abs}^{mean} = 3 \times 10^{-3}$ while the relative error is more scattered with a mean value of $E_{rel}^{mean} = 1 \times 10^{-2}$. Finally, the absolute error distribution for the real and imaginary part are shown in Fig. 6.15: in this case the fit for the imaginary part is slightly worse, with a mean value of 5×10^{-3}.

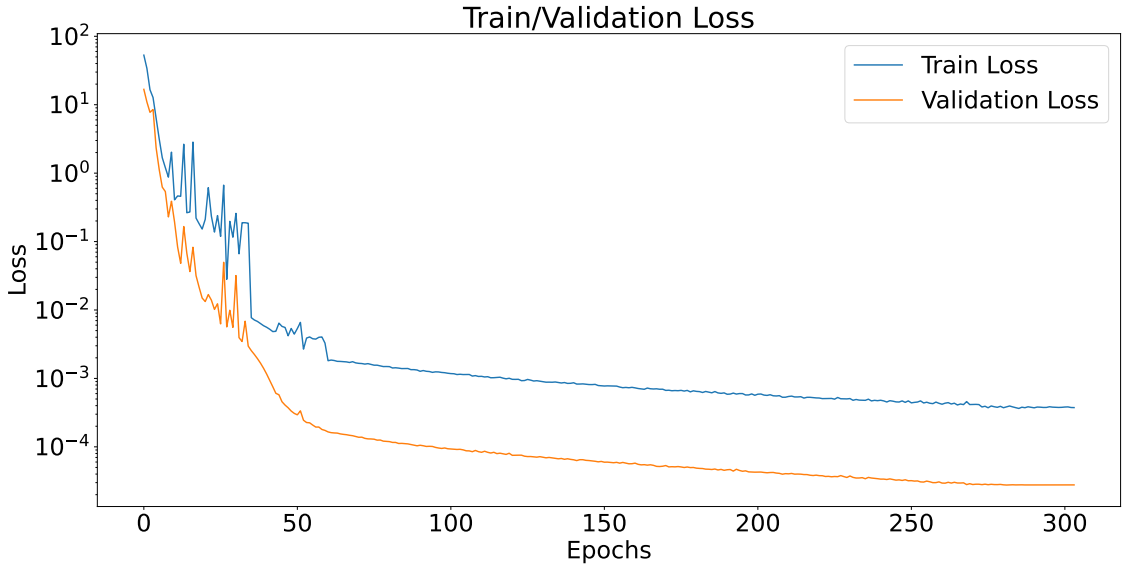


Figure 6.13: Training and test loss over different epochs for the one-mass double Box diagram.

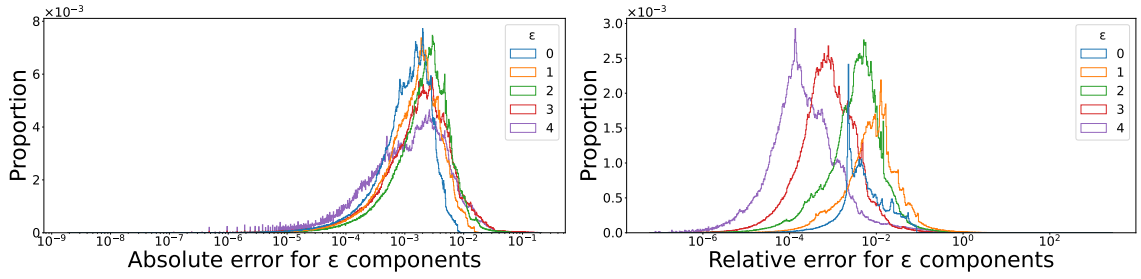


Figure 6.14: (Left panel) Absolute error distribution and (Right panel) relative error distribution for different orders of the Master Integrals of the one-mass double Box diagram

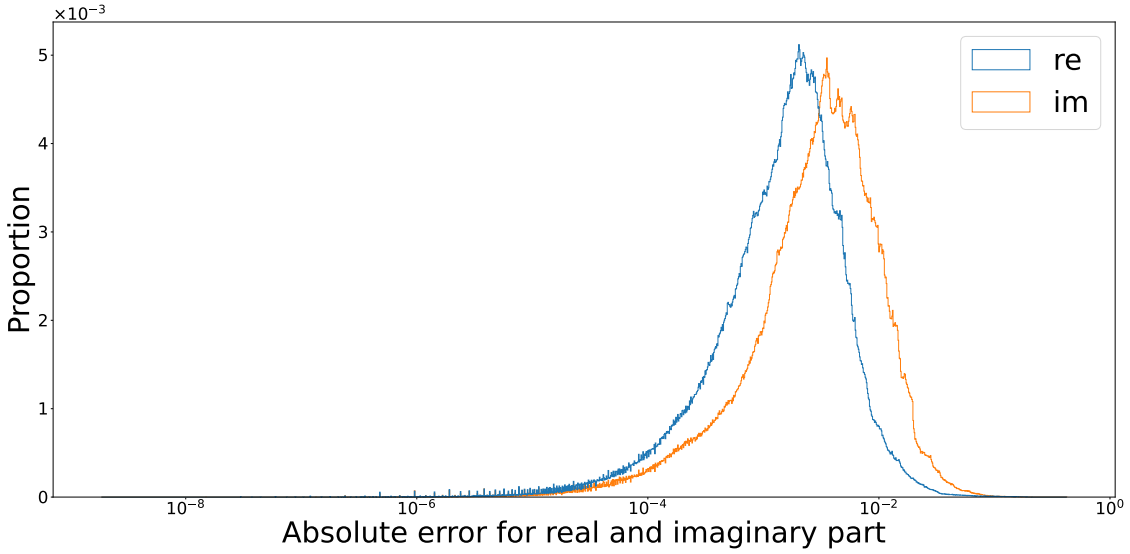


Figure 6.15: Absolute error distribution for the real and imaginary part of the one-mass double Box diagram.

6.3.2 Equal-masses Sunrise

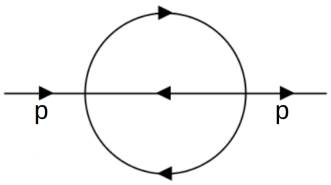


Figure 6.16: Diagram representing the Sunrise Integral family

In this section we deal with the equal-masses Sunrise, this diagram plays a very important role in the evaluation of higher-order corrections in quantum electrodynamics (QED), the electroweak theory and quantum chromodynamics (QCD) and therefore ample literature deals with its computation. There is only one external momentum p whose squared value defines with the mass scale the kinematics invariants:

$$s = p^2, \quad s_m = m^2, \tag{6.47}$$

setting the mass scale we can select s as independent variable for the PINN model, the integrals family is defined by:

$$I_{n_1,n_2,n_3,n_4,n_5}(s,\varepsilon) = \int \frac{d^d k_1}{i\pi^{\frac{D}{2}}} \frac{d^d k_2}{i\pi^{\frac{D}{2}}} \frac{\mu^{4-D}}{D_1^{n_1} D_2^{n_2} D_3^{n_3} D_4^{n_4} D_5^{n_5}} \tag{6.48}$$

with the propagators:

$$\begin{aligned} &D_1 = k_1^2 - m^2, \quad D_2 = k_2^2 - m^2, \quad D_3 = (p + k_1 + k_2)^2 - m^2, \quad D_4 = (k_1 - p)^2 \\ &D_5 = (k_2 - p)^2, \end{aligned} \tag{6.49}$$

where $\{k_i\}_{i=1,2}$ are the two loops momenta; if only one or two internal lines are massive the Sunrise can be expressed in terms of MPLs, however when all of propagators has a mass scale, as in this example, its analytic expression requires elliptic integrals: the two-loop diagram can be defined using the complete elliptic integral of the first and second kind mentioned in Eq. (5.7).
LiteRed reduction generates three Master Integrals:

$$\vec{M}(s,\varepsilon) = \{I_{1,1,0,0,0};\ \ I_{1,1,1,0,0};\ \ I_{2,1,1,0,0}\} \tag{6.50}$$

however, the first MI actually belong to the subtopology and represents a self-energy diagram, the differential equation system for $\vec{M}$ is:

$$\frac{d\vec{M}}{ds} = \mathbb{A}_s \cdot \vec{M} \tag{6.51}$$

with $\mathbb{A}_s$ defined in A.10, we fit for 3 orders in ε so the NN output has $N_M \times N_\varepsilon = 9$ components; the Connection Matrix has 3 orders as well and its denominators vanish for:

$$s = 0, \;\; s = 3, \;\; s = 9, \;\; s = 27, \tag{6.52}$$

the first is a physical singularity while the others are spurious singularities, in our training we work for $s < 0$, so we avoid them alltogether.
The boundary dataset consists of two points the edge of the training phase space region and the testing dataset has 10^4 points uniformly distributed.
The specifics of PINN training are summarized in the tables 6.4:

Hyperparameter	Value
Neural Network	
input size	1
output size	9
hidden layers	[128, 128, 128, 128]
Activation function	GELU
Optimizer	
Algorithm	Adam
Scheduler	ReduceOnPlateau
Initial lr	$1 \cdot 10^{-3}$
factor	0.5
patience	10
cooldown	0
Minimum lr	$1 \cdot 10^{-15}$

Hyperparameter	Value
Sampling	
Base points size	$3.5 \cdot 10^5$
Samples per partition	262144
Partition size	2048
Batch size	128
Partition	1
Advanced options	
ReLoBRaLoss	True
τ	0.5
α	0.99

Table 6.4: Neural Network architecture for the Sunrise diagram.

6.3.2.1 Training result

The training lasted 200 epochs and its results are shown in the plots at the end of this section.
The testing and validation losses shown in Fig. 6.17 have a rapid decreased during training up to values of $\mathcal{O}(10^{-8})$ for L_{train} and $\mathcal{O}(10^{-9})$ for L_{val}.
The mean of the absolute error is $E_{abs}^{mean} = 2 \times 10^{-5}$, its distribution in Fig. 6.18 (left panel) shows a increase as the ε-order increases, finally the relative error distribution is shows in Fig. 6.18 (right panel) and its mean value is $E_{rel}^{mean} = 2 \times 10^{-5}$.
The PINN model is even more successful in fitting the imaginary part, as shown in Fig. 6.19, with a mean absolute error of 9×10^{-8}.

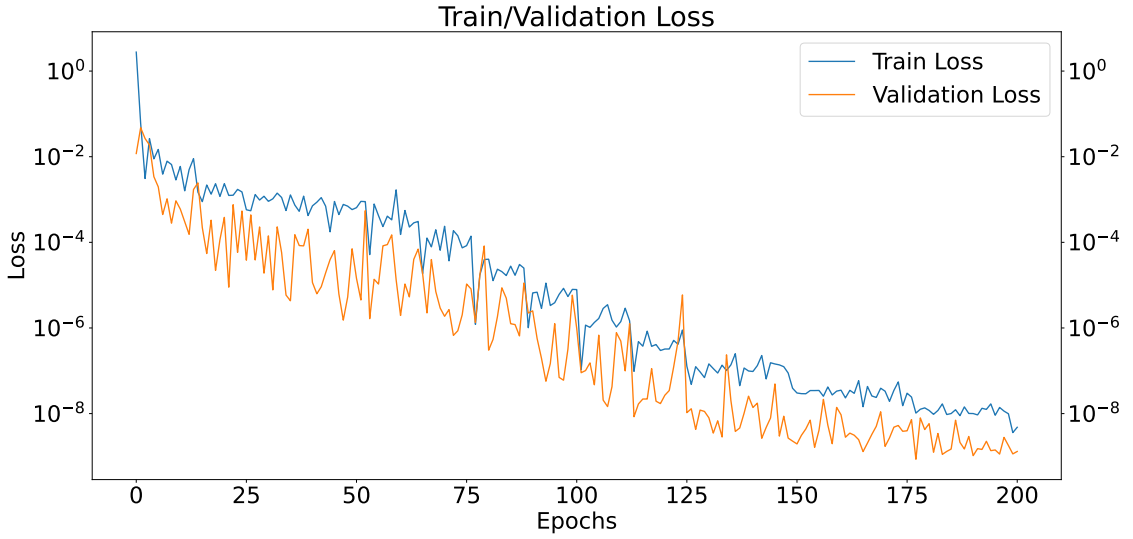


Figure 6.17: Training and test loss over different epochs for the Sunrise diagram.

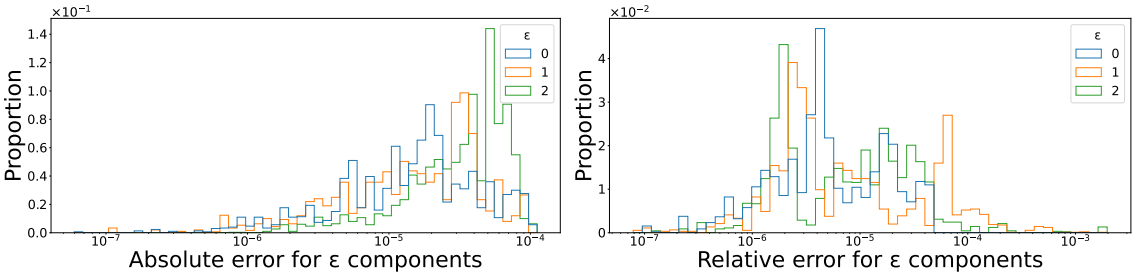


Figure 6.18: (Left panel) Absolute error distribution and (Right panel) relative error distribution for different orders of the Master Integrals of the Sunrise diagram

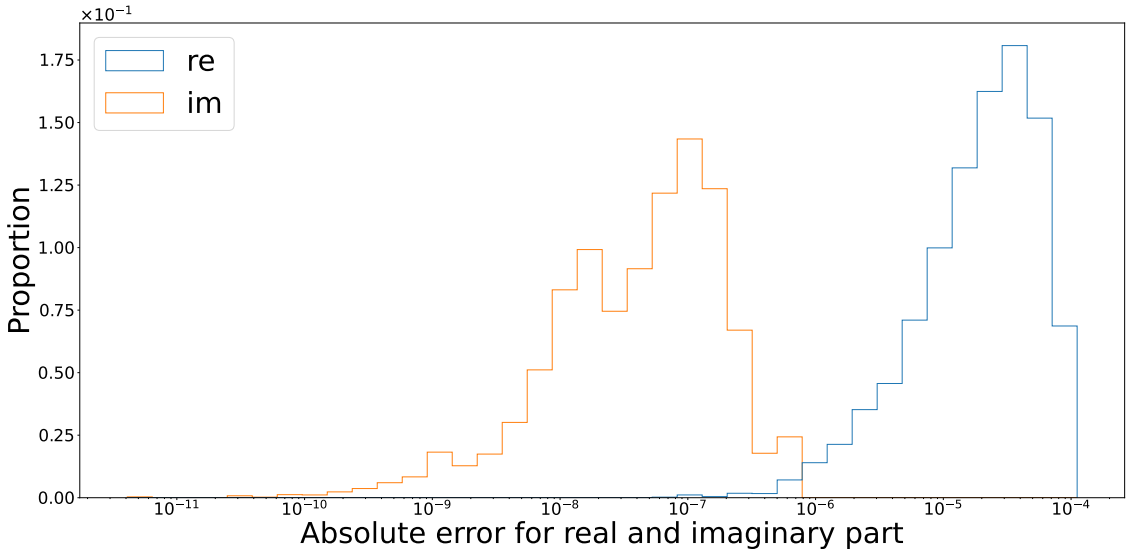


Figure 6.19: Absolute error distribution for the real and imaginary part of the Sunrise diagram.

6.3.3 Non-planar three-point diagram with a massive loop

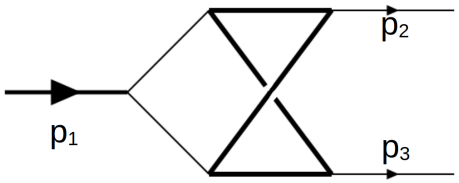


Figure 6.20: Diagram representing the Non planar three points Integral family

The Diagram studied in this section defines a three-point non-planar topology which is relevant for the two-loop QCD corrections to the processes of $t\bar{t}$ production and $\gamma\gamma$ production in gluon fusion:

$$gg \to t\bar{t} \quad gg \to \gamma\gamma \tag{6.53}$$

through a massive top loop. Of the three external momenta one is off-shell and defines a Mandelstam variable:

$$p_2^2 = p_3^2 = 0, \quad p_1^2 = (p_2 + p_3)^2 = s, \tag{6.54}$$

therefore the integrals depend on one variable, for the PINN implementation we use the dimensionless ratio $x = -\frac{s}{m^2}$ The integral representation of the Feynman integrals family is:

$$I_{n_1,n_2,n_3,n_4,n_5,n_6,n_7}(x,\varepsilon) = \int \frac{d^D k_1}{i\pi^{\frac{D}{2}}} \frac{d^D k_2}{i\pi^{\frac{D}{2}}} \frac{\mu^{4-D}}{D_1^{n_1} D_2^{n_2} D_3^{n_3} D_4^{n_4} D_5^{n_5} D_6^{n_6} D_7^{n_7}} \tag{6.55}$$

where the propagators are defined as:

$$\begin{aligned} &D_1 = (k_1 - p_2)^2, \;\; D_2 = (k_2 - p_2)^2 - m^2, \;\; D_3 = (k_1 + p_3)^2, \;\; D_4 = (k_1 - k_2 + p_3)^2 - m^2, \\ &D_5 = (k_1 - k_2)^2 - m^2, \;\; D_6 = k_2^2 - m^2, \;\; D_7 = k_1^2, \end{aligned} \tag{6.56}$$

with k_1,k_2 the loop momenta and, to limit ourselves at the triangle topology as described by Fig. 6.20, only negative powers of the last propagator are admitted $n_7 < 0$. The reduction of the integration by parts identities identifies 11 Master

Integrals, which are presented already with a dimensional regularization:

$$\vec{M}(x,\varepsilon) = \{M_1;\ M_2;\ M_3;\ M_4;\ M_5;\ M_6;\ M_7;\ M_8;\ M_9;\ M_{10};\ M_{11}\} = \{$$
$$\varepsilon^2 I_{0,2,0,2,0,0,0};\ \ \varepsilon^2 s I_{2,2,1,0,0,0,0};\ \ \varepsilon^2 s I_{0,2,1,0,2,0,0};\ \ \varepsilon^2\sqrt{s(s-4m^2)}\left[I_{0,2,2,0,1,0,0} + \frac{1}{2} I_{0,2,1,0,2,0,0}\right];$$
$$\varepsilon^3 s I_{1,2,1,1,0,0,0};\ \ \varepsilon^2\sqrt{s(s+4m^2)}\left[I_{2,2,1,1,0,0,0} - \frac{\varepsilon}{2m^2(1+2\varepsilon)} I_{0,2,0,2,0,0,0}\right];$$
$$\varepsilon^3 s I_{1,2,1,1,0,0,0};\ \ \varepsilon^4 s I_{1,1,0,1,1,1,0};\ \ \varepsilon^4 s I_{1,1,1,1,1,0,0};$$
$$\varepsilon^4 s^2 I_{1,1,1,1,1,1,0};\ \ \varepsilon^4 \frac{s^2(s+16m^2)}{2(1+2\varepsilon)} I_{1,2,1,1,1,1,0}\} \tag{6.57}$$

The first 9 Master Integral in Eq. (6.57) belongs to the subtopologies of the integrals family while M_{10} and M_{11} are the MIs of the top topology, moreover while the entire subtopology can be expressed with multipolylogarithms, the top Master Integrals contain elliptic integrals. This suggest a variation of the procedure used until now by employing a bottom up approach: instead of solving a differential equations system for all the Master integrals, the subtopology is solved first and its analytic expression defines an inhomogeneous term in the DE system of the top MIs which is the subject of this study:

$$\frac{d}{dx}\begin{pmatrix} M_{10} \\ M_{11} \end{pmatrix} = \mathbb{A}_x(x)\begin{pmatrix} M_{10} \\ M_{11} \end{pmatrix} + \begin{pmatrix} N_{10}(\varepsilon,x) \\ N_{11}(\varepsilon,x) \end{pmatrix} \tag{6.58}$$

the connection matrix $\mathbb{A}_x$ is linear in ε and it's defined in A.12 while $\{N_i(\varepsilon,x)\}_{i=10,11}$ are the inhomogeneous terms defined in A.14 and A.15.
The MIs (as well as the inhomogeneous matrix terms) are expanded for 5 orders in ε starting from $l_{min} = -4$ up to ε^0 so that model output then has $N_M \times N_\varepsilon = 10$ components, for their analytic expressions we refer to [42]; finally the connection matrix has two orders in ε.
Looking at the explicit form of the matrices we find possible singularities at:

$$x = -4,\ \ x = 0,\ \ x = 4,\ \ x = 16\,. \tag{6.59}$$

The only physical singularity is for $x = 0$, the rest are spurious, in order to avoid them we work in the range:

$$-20 \leq x \leq c_x \tag{6.60}$$

with the cut $c_x = 10^{-1}$ and avoiding the point $x = -4$ in the sampling of collocation point to avoid the spurious singularity in the connection matrix.
The boundary dataset consists of two points at the edges of the phase space region and the testing datasets has 10^4 points.
The specifics of PINN training are summarized in the tables 6.5:

Hyperparameter	**Value**
Neural Network	
input size	1
output size	10
hidden layers	[128, 128, 128, 128]
Activation function	GELU
Optimizer	
Algorithm	Adam
Scheduler	ReduceOnPlateau
Initial lr	$1 \cdot 10^{-3}$
factor	0.5
patience	10
cooldown	0
Minimum lr	$1 \cdot 10^{-15}$

Hyperparameter	**Value**
Sampling	
Base points size	$2 \cdot 10^5$
Samples per partition	65536
Partition size	512
Batch size	128
Partition	1
Advanced options	
ReLoBRaLoss	True
τ	1
α	0.99

Table 6.5: Neural Network architecture for the non-planar three points diagram.

6.3.3.1 Training result

The training lasted 350 epochs and its results are shown in the plots at the end of this section.
The training and validation loss are shown in Fig. 6.21: while the training loss quickly stops to improve with a final value of $\mathcal{O}(10^{-5})$ the validation loss keeps a decreasing trend up to $\mathcal{O}(10^{-7})$. The absolute error plot in Fig. 6.22 (left panel) reveals a mean value of $E_{abs}^{mean} = 3 \times 10^{-4}$ with a maximum value of $\mathcal{O}(10^{-3})$ while the relative error is showed in Fig. 6.22 (right panel), here only the ε^4 component is shown since it's the only non-zero function, with a mean value of $E_{rel}^{mean} = 4 \times 10^{-4}$. Finally Fig. 6.23 shows the absolute error distribution for both the real and imaginary part, the latter has a mean value of 4×10^{-4}.

Figure 6.21: Training and test loss over different epochs fot the non-planar three points diagram.

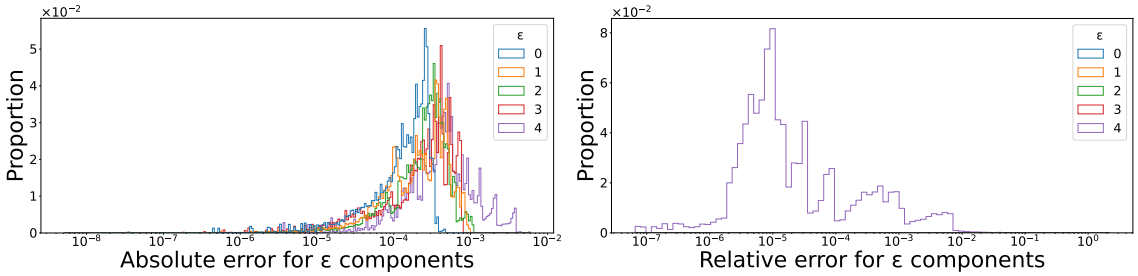


Figure 6.22: (Left panel) Absolute error distribution and (Right panel) relative error distribution for different orders of the Master Integrals of the non-planar three points diagram

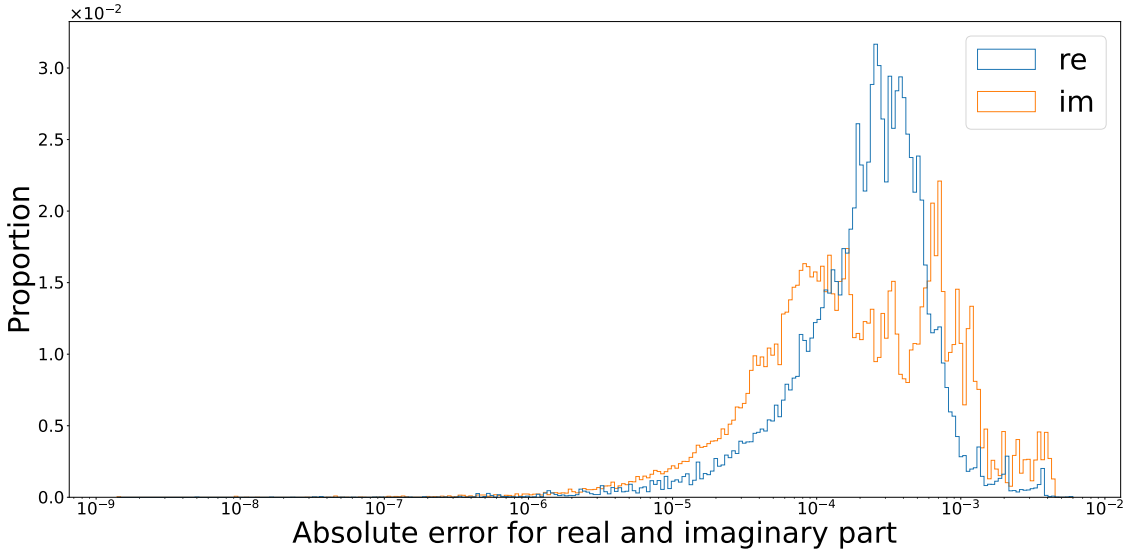


Figure 6.23: Absolute error distributions oh the real and imaginary part for the non-planar three points diagram.

7

Conclusion

In this thesis, we elaborated on the idea of application of Physics-Informed Neural Networks to solving the differential equations that govern Feynman integrals. Feynman integrals are key components in the computation of Scattering Amplitudes in perturbative Quantum Field Theory, and the developments of techniques for their evaluation have been triggering important advances in Physics and Math.

We began this work by providing a thorough introduction to Feynman integrals, tracing their origins from Feynman diagrams and delving into the mathematical intricacies involved in their computation. Our discussion included the process of tensorial decomposition and the reduction of scalar loop integrals to a limited set of independent integrals, known as master integrals, through the use of linear identities, dubbed Integration By Parts identities. These integrals are very difficult to evaluate, particularly in multi-loop diagrams that involve multiple scales and intricate singularities. We reviewed the basic ideas behind one of the most effective method currently available for evaluating these integrals, which involves solving the differential equations satisfied by the Master Integrals, which span the vector space of the scalar Feynman integrals.

Subsequently, following a recent proposal appeared in the literature, we introduced

PINNs, as a promising tool to approximate the solutions of the partial differential equations satisfied by the Master Integrals. Unlike standard neural networks, PINNs incorporate governing physical laws—expressed as differential equations—directly into the training process. This approach allows the network to learn not just from data, but also from the underlying physics, resulting in solutions that have a higher accuracy.

We first applied PINNs to classical differential equations, such as those describing the harmonic oscillator, and, later presenting a novel, original application to hypergeometric functions, addressing the case of functions of increasing complexity, from polynomial to elliptic functions. These initial case studies demonstrated the potential of PINNs to solve classical (systems of) differential equations with high accuracy and efficiency as well as the support higher order differential equations can provide once embedded in PINNs models.

Building on this success, we extended the application of PINNs to the systems of differential equations associated with one- and two-loop Feynman integrals, considering the one-loop massless box diagram, the one-mass box diagram, the one-mass double box diagram, the equal-masses sunrise and the two-loop non-planar elliptic three-point integrals. The latter two applications constitute part of the novel contributions of this thesis.

Our results showed that PINNs could accurately approximate the solutions of these integrals, achieving a high degree of precision.

As novel original proposal to algorithmic developments of PINNs, in order to enhance their effectiveness, we introduced a customizable sampling method for the training dataset, improving the selection of the collocation points. We also implemented a loss balancing scheme, known as Relative Loss Balancing with Random Lookbacks, which automatically adjusts the scalar multipliers for different loss functions. Additionally, we generalized the system of differential equations that PINNs can handle by including the possibility of inhomogeneous terms, further broadening the applicability of this method.

Looking forward, the application of PINNs to higher-loop Feynman integrals—which involve even more complex and computationally demanding systems of differential equations—represents an exciting frontier in this research. Develop-

ing new strategies for training PINNs on these challenging problems, potentially through hybrid approaches that combine traditional methods with Machine Learning, or by optimizing the selection of collocation points, could lead to significant advancements in our ability to compute Scattering Amplitudes in Quantum Field Theory.

Through this exploration, we highlight the possible powerful synergy between Machine Learning and Feynman calculus, suggesting possible advancements in the area of Computational Quantum Field Theory, and more generally for problems in Applied Mathematics and Computational Science benefitting from the development of new classes of numerical solvers for PDEs, which can be applied in modern Physics and Mathematical applications. Eventually for Feynman integrals, but more generally to the larger class of special functions admitting a Twisted Period Integral representation - currently under vivid investigation in Theoretical Physics. The insights gained from this work lay a solid foundation for future research that will continue to push the boundaries of what is possible in this rapidly evolving field.

A

Connection Matrices of the examples

Here we present kinematic invariants for each example together with the connection matrices used during training

A.1 Massless Box

The kinematic invariants are:

$$s_0 = (p_1 + p_2)^2, \quad s_1 = (p_2 + p_3)^2 \tag{A.1}$$

and the connection matrix for $x = \frac{s_1}{s_0}$ is:

$$\mathbb{A}_x = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{\varepsilon}{x} & 0 \\ \frac{2(1-2\varepsilon)}{1+x} & \frac{2(-1+2\varepsilon)x}{1+x} & \frac{1+x+\varepsilon x}{x(1+x)} \end{pmatrix} \tag{A.2}$$

A.2 One-Mass Box

The kinematic invariants are:

$$s_0 = (p_1 + p_2)^2, \;\; s_1 = (p_2 + p_3)^2, \;\; s_2 = m^2, \tag{A.3}$$

The connection matrices are:

$$\mathbb{A}_{s_1} = \begin{pmatrix} \frac{\varepsilon(-s_0+s_2)}{s_1(s_0+s_1-s_2)} & \frac{2(1-2\varepsilon)s_0}{(s_1-s_2)(s_0+s_1-s_4)} & \frac{-2(1-2\varepsilon)s_2}{(s_1-s_2)(s_0+s_1-s_4)} & \frac{2(1-2\varepsilon)s_2^2}{s_0 s_1(-s_0-s_1+s_2)} \\ 0 & -\frac{\varepsilon}{s_1} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{A.4}$$

$$\mathbb{A}_{s_2} = \begin{pmatrix} \frac{s_0+s_1-s_2+\varepsilon(s_0+s_1-2s_2)}{s_2(s_0+s_1-s_2)} & \frac{-2(1-2\varepsilon)s_0}{(s_1-s_2)(s_0+s_1-s_2)} & \frac{-2(1_2\varepsilon)s_1(s_0+s_1-2s_2)}{(s_2-s_0)(s_1-s_2)(s_0+s_1-s_2)} & \frac{-2(1-2\varepsilon)s_2^2}{s_0(s_2-s_0)(s_2-s_0-s_1)} \\ 0 & \frac{1+\varepsilon}{s_2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\varepsilon-1}{s_2} \end{pmatrix} \tag{A.5}$$

A.3 One-Mass Double Box

The kinematic invariants are:

$$s_0 = (p_1 + p_2)^2, \;\; s_1 = (p_2 + p_3)^2, \;\; s_m = m^2, \tag{A.6}$$

To simplify the expression of the connection matrix we define the collection of variables:

$$\begin{aligned}
x = \{ & s_1, \;\; s_0 + s_1, \;\; s_m, \;\; -s_1 + s_m, \;\; \frac{1}{(-s_1 + s_m)}, \;\; s_0 + s_1 - s_m, \;\; \frac{1}{(s_0 - s_m)}, \;\; \frac{1}{(-s_0 + s_m)}, \\
& s_1 - s_m, \;\; \frac{1}{(s_1 - s_m)}, \;\; \frac{1}{(s_0 + s_1 - s_m)}, \;\; \frac{1}{s_m}, \;\; \frac{1}{s_1}, \;\; \frac{1}{(-s_0 - s_1 + s_m)}, \;\; s_0, \\
& -s_0 + s_m, \;\; s_0 - s_m, \;\; 28s_0 + 21s_1 - 19s_m, \;\; \frac{1}{(s_0^2 + s_0 s_1 - s_0 s_m)}, \;\; \frac{-8}{(s_0 + s_1 - s_m)} + \frac{3}{s_1}, \\
& s_0 + s_1 + 2s_m, \;\; -10s_0 - 9s_1 + 7s_m, \;\; 2s_0 + s_1 - s_m, \;\; 4s_0 + 3s_1 - 4s_m, \\
& s_1^2 + s_1 s_m + 2s_m(s_0 - s_m), \;\; s_1 + s_m, \;\; 4s_0^2 + 13s_0 s_1 + 4s_0 s_m + 10s_1^2 - 3s_1 s_m - 8s_m^2, \\
& 2s_0(s_1 - 2s_m) + (s_1 - s_m)(3s_1 - 4s_m), \;\; -s_0 s_m + s_m^2, \;\; -s_0 s_1 + s_0 s_m, \;\; 2s_0 s_1^2 s_m - 2s_0 s_1 s_m^2, \\
& 2s_0 + s_1 - 2s_m, \;\; s_0 + 2s_1 - s_m, \;\; 8s_0^2 + 15s_0 s_1 + 6s_1^2 + 4s_m^2 - 3s_m(4s_0 + 5s_1), \\
& 4s_0^2 + 9s_0 s_1 - 6s_0 s_m + 6s_1^2 - 9s_1 s_m + 2s_m^2, \;\; -8s_0 + 4s_1, \;\; 2s_0 + 3s_1, \\
& 40s_0^2 + 69s_0 s_1 - 48s_0 s_m + 30s_1^2 - 57s_1 s_m + 8s_m^2, \\
& 8s_0^3 + 16s_0^2 s_1 + 13s_0 s_1^2 + 6s_1^3 - 4s_m^3 + 7s_m^2(2s_0 + 3s_1) - s_m(18s_0^2 + 35s_0 s_1 + 21s_1^2), \\
& -(s_0^2)s_m - s_0 s_1(3s_0 + 4s_1) + s_m^2(s_0 + s_1), \;\; 7s_0^2 + 13s_0 s_1 - 9s_0 s_m + 6s_1^2 - 11s_1 s_m \\
& + 2s_m^2, \;\; 8s_0 + 3s_1 - 8s_m, \\
& -4s_0 - 3s_1 + 4s_m, \;\; s_1^2 - s_1 s_m, \;\; s_0 s_1^2 - s_0 s_1 s_m, \;\; 40s_0 + 21s_1 - 40s_m, \\
& 8s_0^2 + 6s_0 s_1 - 16s_0 s_m + 3s_1^2 - 9s_1 s_m, \;\; +8s_m^2, \;\; 2s_0 + 3s_1 - 2s_m, \;\; s_0 s_1 + s_1^2, \\
& -2s_0 - 3s_1 + 2s_m, \;\; 8s_0 + 9s_1 - 8s_m, \;\; \frac{1}{(s_0 s_1 s_m + s_1^2 s_m)}, \;\; -2s_0 - 2s_1 + s_m, \\
& \frac{-19}{(s_0 + s_1 - s_m)} + \frac{4}{(s_0 - s_m)} - \frac{10}{s_m}, \;\; 8s_0(s_0 + s_1) - 7s_m^2 - s_m(s_0 + 10s_1), \\
& 16s_0(s_0 + s_1) - 2s_m^2 - s_m(14s_0 + 19s_1), \;\; s_0 + s_1 + s_m, \\
& s_1(s_0 + s_1) + 4s_m^2 + s_m(2s_0 + s_1), \;\; \frac{-7}{(s_0 + s_1 - s_m)} + \frac{1}{(s_0 - s_m)} - \frac{4}{s_m}, \;\; 2s_0 + 2s_1 - 3s_m, \\
& -8s_0(s_0 + s_1) - 2s_m^2 + 10s_m(s_0 + s_1), \;\; -86s_0(s_0 + s_1) + 13s_m^2 + s_m(73s_0 + 100s_1),
\end{aligned}$$

$$
\begin{aligned}
&s_0^2 + s_0(s_1 + s_m) + s_m(s_1 - 2s_m),\ \ s_0^2 - s_0 s_m,\\
&16s_0^2 + 28s_0 s_1 - 2s_0 s_m + 12s_1^2 - 5s_1 s_m - 5s_m^2,\ \ -12s_0(s_0 + s_1) + 3s_m(4s_0 + 5s_1),\\
&12s_0(s_0 + s_1) - 3s_m(4s_0 + 5s_1),\ \ \frac{-9s_0}{(s_1 s_m)} + \frac{7}{(s_0 + s_1 - s_m)},\\
&\frac{8}{(s_0 + s_1 - s_m)} + \frac{1}{(s_0 - s_m)} + \frac{6}{s_m},\ \ 24s_0(s_0 + s_1) - 6s_m(4s_0 + 3s_1),\\
&\frac{-1}{s_m} + \frac{(-3s_0 - 3s_1)}{(s_1(s_0 + s_1 - s_m))},\ \ \frac{-3s_0}{(s_1 s_m)} + \frac{1}{(s_0 + s_1 - s_m)},\ \ \frac{1}{(-s_0 + s_m)} + \frac{(-6s_0 - 6s_1)}{(s_1 s_m)},\\
&-63s_0(s_0 + s_1) + s_m(63s_0 + 47s_1),\ \ 12s_0^2 + 21s_0 s_1 - 9s_0 s_m + 9s_1^2 - 7s_1 s_m,\\
&-18s_0(s_0 + s_1) + 6s_m(3s_0 + 2s_1),\ \ 9s_0(s_0 + s_1) - 3s_m(3s_0 + 2s_1),\ \ s_0 s_m - s_m^2,\\
&s_0 + s_1 - 4s_m,\ \ \frac{-1}{(s_1 s_m)} - \frac{1}{(s_0(s_0 + s_1 - s_m))},\ \ \frac{-1}{(s_1(s_0 + s_1 - s_m))} - \frac{1}{(s_0 s_m)},\\
&-3s_0^2 - 2s_0 s_1 + 8s_0 s_m + 4s_1 s_m - 5s_m^2,\ \ s_0 s_1 + 3s_m^2 - 3s_m(s_0 + s_1),\ \ 3s_0 + 4s_1 - 3s_m,\\
&-s_0 s_1 - 3s_m^2 + 3s_m(s_0 + s_1),\ \ 3s_0^2 + 2s_0(s_1 + 8s_m) + s_m(18s_1 - 19s_m),\\
&-s_0(s_1 + s_m) + s_m(-s_1 + s_m),\ \ s_0(s_1 + s_m) + s_m(s_1 - s_m),\ \ s_0 s_1 s_m - s_0 s_m^2,\\
&s_0 + s_1 - 2s_m,\ \ 2s_0 s_1 s_m^2 - 2s_0 s_m^3,\ \ s_0(s_1 + s_m) + s_1^2 - s_1 s_m - s_m^2,\\
&4s_0(s_1 + s_m) + (s_1 - 2s_m)(3s_1 + 2s_m),\ \ 2s_0(s_1 + s_m) + 3s_1^2 - 2s_1 s_m - 2s_m^2,\\
&8s_0 - 4s_1,\ \ -20s_0(s_1 + s_m) - 21s_1^2 + 20s_1 s_m + 20s_m^2,\\
&4s_0^2(s_1 + s_m) + 2s_0(s_1 - 2s_m)(3s_1 + 2s_m) + (s_1 - 2s_m)(3s_1^2 - 3s_1 s_m - 2s_m^2),\ \ s_0(s_1 + 3s_m) +\\
&s_m(s_1 - 2s_m),\ \ -s_1(s_0 + s_1) + s_m^2 + s_m(-s_0 + s_1),\ \ 3s_1(s_0 + s_1) - 4s_m^2 - 2s_m(-2s_0 + s_1),\\
&8s_0 + 8s_1 - 13s_m,\ \ 4s_0 + 4s_1 - 5s_m,\ \ s_1 s_m - s_m^2,\ \ 4s_0 + 4s_1 - 5s_m,\\
&-s_0 s_1 s_m + s_0 s_m^2,\ \ -40s_0 - 40s_1 + 59s_m,\ \ 8s_0^2 + 14s_0 s_1 - 24s_0 s_m + 6s_1^2 - 17s_1 s_m + 13s_m^2,\\
&-6s_0 - 6s_1 + 9s_m,\ \ 6s_0 + 6s_1 - 9s_m,\ \ -2s_0 - 3s_1 + 3s_m,\ \ s_1 - 2s_m,\ \ -s_1 s_m + s_m^2\}
\end{aligned}
\tag{A.7}
$$

and we define x_i as the i-th element in this set; the connection matrix is then:

$$
A_{s_1} = \begin{pmatrix}
+\varepsilon x_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{-1}{x_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & +\varepsilon x_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{-1}{x_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & +\varepsilon x_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{-1}{x_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{x_2}{x_3} + \varepsilon^{\frac{x_0}{x_8}} & x_4 + \varepsilon x_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +\varepsilon x_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{x_0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{x_5} + \varepsilon x_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +\varepsilon x_6 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{-1}{x_1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x_4 + \varepsilon x_9 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{-1}{x_1} + \varepsilon x_7 & 0 & 0 & \frac{-1}{x_1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +\varepsilon x_{11} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\tag{A.8}
$$

A.4 Equal mass Sunrise

The kinematic invariants are:

$$s = p^2, \quad s_m = m^2, \tag{A.9}$$

The connection matrix (for $s_m = 3$) is sown in the next page.

$$
\mathbb{A}_s = \begin{pmatrix}
-\frac{2}{3(-27+s)(-3+s)} + \varepsilon\frac{4}{3(-27+s)(-3+s)} - \varepsilon^2\frac{2}{3(-27+s)(-3+s)} & -\frac{2(-9+s)}{(-27+s)(-3+s)s} + \varepsilon\frac{7(-9+s)}{(-27+s)(-3+s)s} - \varepsilon^2\frac{6(-9+s)}{(-27+s)(-3+s)s} & -\frac{162-30s}{(-27+s)(-3+s)s} - \varepsilon\frac{-243+30s+s^2}{(-27+s)(-3+s)s} \\
0 & \frac{1}{s} - \frac{\varepsilon^2}{s} & -\frac{9}{s} \\
0 & 0 & 0
\end{pmatrix}
\tag{A.10}
$$

A.5 Non planar two-loops diagram

The kinematic invariants are:

$$s = (p_1 + p_2)^2, \;\; s_m = m^2, \tag{A.11}$$

The PINN fit is performed w.r.t. the variable $x = -\frac{s}{s_m}$.
The connection matrix is defined as:

$$\mathbb{A}_x(x) = \begin{pmatrix} -\varepsilon\frac{2}{x} & \frac{1}{2\,(x-16)} - \frac{1}{2x} + \varepsilon\left(\frac{1}{x-16} - \frac{1}{x}\right) \\ \\ \frac{1}{2x}(1+\varepsilon) & \frac{1}{x} + \varepsilon\left(\frac{1}{x} - \frac{1}{x-16}\right) \end{pmatrix} \tag{A.12}$$

To simplify the expression of the inhomogenous terms we first defines the functions in x:

$$\begin{aligned}
&\eta_1(x) = \sqrt{x}, \;\; \eta_2(x) = \frac{1}{2}(\sqrt{x} + \sqrt{x+4}), \;\; \eta_3(x) = \sqrt{x+4}, \\
&\eta_4(x) = \frac{1}{2}(\sqrt{x} + \sqrt{x-4}), \;\; \eta_5(x) = \sqrt{x-4}, \\
&\eta_6(x) = \sqrt{-x}, \;\; \eta_7(x) = \frac{1}{2}(\sqrt{-x} + \sqrt{-x-4}), \;\; \eta_8(x) = \sqrt{-x-4}, \\
&\eta_9(x) = \frac{1}{2}(\sqrt{-x} + \sqrt{-x+4}), \;\; \eta_{10}(x) = \sqrt{-x+4},
\end{aligned} \tag{A.13}$$

then the inhomogeneous terms for $-4 < x < 0$ are:

$$\mathbb{N}^{(-4,0)}(x) = \begin{pmatrix} \mathbb{N}_{10}^{(-4,0)}(x) \\ \\ \mathbb{N}_{11}^{(-4,0)}(x) \end{pmatrix} =$$

$$\begin{pmatrix} 0 \\ \\ \varepsilon^4 \left(-5 arcsec^2\left(\frac{2}{\eta_3(x)}\right) + 3\eta_6(x)\frac{\zeta_2 + \ln^2 \eta_9(x) - Li_2\left(\frac{1}{\eta_9^2(x)}\right)}{\eta_{10}(x)} + i\pi\frac{3\eta_6(x)}{\eta_{10}(x)} \ln \eta_9(x) \right) \end{pmatrix} \tag{A.14}$$

and the ones for $-\infty < x < -4$ are:

$$\mathbb{N}^{(-\infty,0)}(x) = \begin{pmatrix} \mathbb{N}_{10}^{(-\infty,0)}(x) \\ \\ \mathbb{N}_{11}^{(-\infty,0)}(x) \end{pmatrix} =$$

$$\begin{pmatrix} 0 \\ \varepsilon^4 \left(5 \ln^2 \eta_7(x) \frac{15}{2} \zeta_2 + 3\eta_6(x) \frac{\zeta_2 + \ln^2 \eta_9(x) - Li_2\left(\frac{1}{\eta_9^2(x)}\right)}{\eta_{10}(x)} \right) + i\pi \left(\frac{3\eta_6(x)}{\eta_{10}(x)} \ln \eta_9(x) - 5 \ln \eta_7(x) \right) \end{pmatrix} \tag{A.15}$$

Bibliography

[1] Gerard t Hooft and M. J. G. Veltman. "Regularization and Renormalization of Gauge Fields". In: *Nucl. Phys. B* 44 (1972), pp. 189–213. doi: 10.1016/0550-3213(72)90279-9. eprint: NUPHA.B44.189.1972.

[2] K. G. Chetyrkin and F. V. Tkachov. "Integration by parts: The algorithm to calculate β-functions in 4 loops". In: *Nucl. Phys. B* 192 (1981), pp. 159–204. doi: 10.1016/0550-3213(81)90199-1.

[3] Pierpaolo Mastrolia and Sebastian Mizera. "Feynman Integrals and Intersection Theory". In: *JHEP* 02 (2019), p. 139. doi: 10.1007/JHEP02(2019)139. arXiv: 1810.03818 [hep-th].

[4] Hjalte Frellesvig et al. "Decomposition of Feynman Integrals on the Maximal Cut by Intersection Numbers". In: *JHEP* 05 (2019), p. 153. doi: 10.1007/JHEP05(2019)153. arXiv: 1901.11510 [hep-ph].

[5] Hjalte Frellesvig et al. "Vector Space of Feynman Integrals and Multivariate Intersection Numbers". In: *Phys. Rev. Lett.* 123.20 (2019), p. 201602. doi: 10.1103/PhysRevLett.123.201602. arXiv: 1907.02000 [hep-th].

[6] Hjalte Frellesvig et al. "Decomposition of Feynman Integrals by Multivariate Intersection Numbers". In: *JHEP* 03 (2021), p. 027. doi: 10.1007/JHEP03(2021)027. arXiv: 2008.04823 [hep-th].

[7] Sebastian Mizera. *Aspects of Scattering Amplitudes and Moduli Space Localization*. 2020. doi: https://doi.org/10.1007/978-3-030-53010-5. arXiv: 1906.02099 [hep-th]. url: https://arxiv.org/abs/1906.02099.

[8] Stefan Weinzierl. "On the computation of intersection numbers for twisted cocycles". In: *J. Math. Phys.* 62.7 (2021), p. 072301. doi: 10.1063/5.0054292. arXiv: 2002.01930 [math-ph].

[9] Simon Caron-Huot and Andrzej Pokraka. "Duals of Feynman Integrals. Part II. Generalized unitarity". In: *JHEP* 04 (2022), p. 078. doi: 10.1007/JHEP04(2022)078. arXiv: 2112.00055 [hep-th].

[10] Simon Caron-Huot and Andrzej Pokraka. “Duals of Feynman integrals. Part I. Differential equations”. In: *JHEP* 12 (2021), p. 045. doi: 10.1007/JHEP12(2021)045. arXiv: 2104.06898 [hep-th].

[11] Vsevolod Chestnov et al. “Intersection numbers from higher-order partial differential equations”. In: *JHEP* 06 (2023), p. 131. doi: 10.1007/JHEP06(2023)131. arXiv: 2209.01997 [hep-th].

[12] Francesco Calisto, Ryan Moodie, and Simone Zoia. “Learning Feynman integrals from differential equations with neural networks”. In: *Journal of High Energy Physics* 2024.7 (July 2024). issn: 1029-8479. doi: 10.1007/jhep07(2024)124. url: http://dx.doi.org/10.1007/JHEP07(2024)124.

[13] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[14] Maziar Raissi and George Em Karniadakis. “Hidden physics models: Machine learning of nonlinear partial differential equations”. In: *Journal of Computational Physics* 357 (Mar. 2018), pp. 125–141. issn: 0021-9991. doi: 10.1016/j.jcp.2017.11.039. url: http://dx.doi.org/10.1016/j.jcp.2017.11.039.

[15] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. 2017. arXiv: 1711.10561 [cs.AI]. url: https://arxiv.org/abs/1711.10561.

[16] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. 2017. arXiv: 1711.10566 [cs.AI]. url: https://arxiv.org/abs/1711.10566.

[17] MARIO ARGERI and PIERPAOLO MASTROLIA. “FEYNMAN DIAGRAMS AND DIFFERENTIAL EQUATIONS”. In: *International Journal of Modern Physics A* 22.24 (Sept. 2007), pp. 4375–4436. issn: 1793-656X. doi: 10.1142/s0217751x07037147. url: http://dx.doi.org/10.1142/S0217751X07037147.

[18] A. G. GROZIN. “INTEGRATION BY PARTS: AN INTRODUCTION”. In: *International Journal of Modern Physics A* 26.17 (July 2011), pp. 2807–2854. issn: 1793-656X. doi: 10.1142/s0217751x11053687. url: http://dx.doi.org/10.1142/S0217751X11053687.

[19] Laporta. "High-precision calculation of multi-loop Feynman integrals by difference equations". In: *International Journal of Modern Physics A* 15 (2000), p. 5087. issn: 0217-751X. doi: 10.1016/s0217-751x(00)00215-7. url: http://dx.doi.org/10.1016/S0217-751X(00)00215-7.

[20] A. V. Smirnov and A. V. Petukhov. *The number of master integrals is finite*. 2010. arXiv: 1004.4199 [hep-th]. url: https://arxiv.org/abs/1004.4199.

[21] A.V. Kotikov. "Differential equations method. New technique for massive Feynman diagram calculation". In: *Physics Letters B* 254.1 (1991), pp. 158–164. issn: 0370-2693. doi: https://doi.org/10.1016/0370-2693(91)90413-K. url: http://www.sciencedirect.com/science/article/pii/037026939190413K.

[22] E. Remiddi. "Differential equations for Feynman graph amplitudes". In: *Il Nuovo Cimento A* 110.12 (Dec. 1997), pp. 1435–1452. issn: 1826-9869. doi: 10.1007/bf03185566. url: http://dx.doi.org/10.1007/BF03185566.

[23] Lorenzo Tancredi. "Methods for Multiloop Computations and their Application to Vector Boson Pair Production in NNLO QCD". PhD thesis. University of Zurich, July 2014. url: https://doi.org/10.5167/uzh-98025.

[24] Johannes M. Henn. "Multiloop Integrals in Dimensional Regularization Made Simple". In: *Physical Review Letters* 110.25 (June 2013). issn: 1079-7114. doi: 10.1103/physrevlett.110.251601. url: http://dx.doi.org/10.1103/PhysRevLett.110.251601.

[25] Mario Argeri et al. "Magnus and Dyson series for Master Integrals". In: *Journal of High Energy Physics* 2014.3 (Mar. 2014). issn: 1029-8479. doi: 10.1007/jhep03(2014)082. url: http://dx.doi.org/10.1007/JHEP03(2014)082.

[26] Xiao Liu and Yan-Qing Ma. "AMFlow: A Mathematica package for Feynman integrals computation via auxiliary mass flow". In: *Computer Physics Communications* 283 (Feb. 2023), p. 108565. issn: 0010-4655. doi: 10.1016/j.cpc.2022.108565. url: http://dx.doi.org/10.1016/j.cpc.2022.108565.

[27] Xiao Liu, Yan-Qing Ma, and Chen-Yu Wang. "A systematic and efficient method to compute multi-loop master integrals". In: *Physics Letters B* 779 (Apr. 2018), pp. 353–357. issn: 0370-2693. doi: 10.1016/j.physletb.2018.02.026. url: http://dx.doi.org/10.1016/j.physletb.2018.02.026.

[28] Sara Brown. *Machine Learning, explained*. 2021. url: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (visited on 04/21/2021).

[29] Luis Medrano Navarro, Luis Martín Moreno, and Sergio G Rodrigo. *Solving differential equations with Deep Learning: a beginner's guide*. 2023. arXiv: 2307.11237 [physics.comp-ph]. url: https://arxiv.org/abs/2307.11237.

[30] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Math. Control Signal Systems* 2 (1989), pp. 303–314. doi: https://doi.org/10.1007/BF02551274.

[31] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. issn: 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. url: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[32] Michael A. Nielsen. *Neural Networks and Deep learning*. Determination Press, 2015.

[33] Ronen Eldan and Ohad Shamir. *The Power of Depth for Feedforward Neural Networks*. 2016. arXiv: 1512.03965 [cs.LG]. url: https://arxiv.org/abs/1512.03965.

[34] Atilim Gunes Baydin et al. *Automatic differentiation in machine learning: a survey*. 2018. arXiv: 1502.05767 [cs.SC]. url: https://arxiv.org/abs/1502.05767.

[35] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. url: https://arxiv.org/abs/1412.6980.

[36] Remco van der Meer, Cornelis Oosterlee, and Anastasia Borovykh. *Optimally weighted loss functions for solving PDEs with Neural Networks*. 2021. arXiv: 2002.06269 [math.NA]. url: https://arxiv.org/abs/2002.06269.

[37] George Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* (May 2021), pp. 1–19. doi: 10.1038/s42254-021-00314-5.

[38] Rafael Bischof and Michael Kraus. "Multi-Objective Loss Balancing for Physics-Informed Deep Learning". In: *arXiv e-prints*, arXiv:2110.09813 (Oct.

2021), arXiv:2110.09813. doi: 10.48550/arXiv.2110.09813. arXiv: 2110.09813 [cs.LG].

[39] Hubert Baty and Leo Baty. *Solving differential equations using physics informed deep learning: a hand-on tutorial with benchmark tests*. 2023. arXiv: 2302.12260 [cs.LG]. url: https://arxiv.org/abs/2302.12260.

[40] Siddharth Rout, Vikas Dwivedi, and Balaji Srinivasan. *Numerical Approximation in CFD Problems Using Physics Informed Machine Learning*. 2021. arXiv: 2111.02987 [cs.LG]. url: https://arxiv.org/abs/2111.02987.

[41] Eric Aislan Antonelo et al. "Physics-informed neural nets for control of dynamical systems". In: *Neurocomputing* 579 (Apr. 2024), p. 127419. issn: 0925-2312. doi: 10.1016/j.neucom.2024.127419. url: http://dx.doi.org/10.1016/j.neucom.2024.127419.

[42] Andreas von Manteuffel and Lorenzo Tancredi. "A non-planar two-loop three-point function beyond multiple polylogarithms". In: *Journal of High Energy Physics* 2017.6 (June 2017). issn: 1029-8479. doi: 10.1007/jhep06(2017)127. url: http://dx.doi.org/10.1007/JHEP06(2017)127.

[43] S. Borowka et al. "pySecDec: A toolbox for the numerical evaluation of multi-scale integrals". In: *Computer Physics Communications* 222 (Jan. 2018), pp. 313–326. issn: 0010-4655. doi: 10.1016/j.cpc.2017.09.015. url: http://dx.doi.org/10.1016/j.cpc.2017.09.015.

[44] G. Heinrich et al. "Numerical scattering amplitudes with pySecDec". In: *Computer Physics Communications* 295 (Feb. 2024), p. 108956. issn: 0010-4655. doi: 10.1016/j.cpc.2023.108956. url: http://dx.doi.org/10.1016/j.cpc.2023.108956.

[45] Angelos Katharopoulos and François Fleuret. *Not All Samples Are Created Equal: Deep Learning with Importance Sampling*. 2019. arXiv: 1803.00942 [cs.LG]. url: https://arxiv.org/abs/1803.00942.

[46] Mohammad Amin Nabian, Rini Jasmine Gladstone, and Hadi Meidani. "Efficient training of physics-informed neural networks via importance sampling". In: *Computer-Aided Civil and Infrastructure Engineering* 36.8 (Apr. 2021), pp. 962–977. issn: 1467-8667. doi: 10.1111/mice.12685. url: http://dx.doi.org/10.1111/mice.12685.

Acknowledgments

First of all I wish to express my deepest gratitude my supervisor and co-supervisor, Dr. Manoj Kumar Mandal and Prof. Pierpaolo Mastrolia for the trust they have placed in me despite logistical difficulties, for all the time and the invaluable dedication they spent in this project and for the passion and important lessons they passed on to me.
I also wish to thank Prof. Alessandro Sperduti and his team at the Department of Mathematics to make time for a meeting and for the insight and expertise they provided on the theory of Neural Networks.
I am grateful to the INFN institute in Padua for its willingness to provide access to its Virtual Machine services.
Finally I wish to thank my family and in particular my sister Francesca, without whose support this work would not have been possible.