Durham University Department of Computer Science

Academic year of 2021 MSc Dissertation submitted for the degree of
MSc Scientific Computing & Data Analysis

Parametric neural network integration for Feynman integrals.

Student Author:
Minwoo Kim

Supervisor:
Prof. Daniel Maitre
Prof. Stephen Jones

Abstract

This paper investigates the new approach to the parametric integration of functions on an n-dimensional hyper-cube. Many subject areas use the hyper-cube function while changing parameters. If the function is too complex, it takes considerable time to calculate for many parameters. Hence, a new approach to parametric integration enable us to reduce the needed time and resources. In high-energy particle physics, the function of the n-dimensional hyper-cube has been applied to the sector decomposed Feynman parametrised function, which is the disassembled part of the Feynman integral in the Feynman diagram. Because it is extremely difficult to calculate the decomposed Feynman parametrised integral in an analytical way, It is quite useful to test and implement a new numerical integration method. The most arduous aspect of calculating the Feynman integrals is the infinite possible outputs created from the collision of the particles in various attribute values. Therefore, this paper uses the parametric integral of artificial neural networks, which aim to reduce the integration time of a function with manifold parameters through a one-time trained network. The Feynman integral is an excellent case for testing parametric integration because of its infinite possibility. Furthermore, it was the first time an artificial neural network was used to solve a parametric integral.

This forefront technique will introduce a new direction to reducing compute resources and time in calculating integrals. Furthermore, this method is expected to contribute to academic and industrial fields that measure the integration of multi-dimensional functions with many parameters.

Keywords : Artificial neural network - Feynman integral - Approximation - Parametric integration

1 Introduction

1.1 Case study and organization of paper

Previous research by Lloyd [1] introduced a numerical integration method for approximated functions using artificial neural networks. The study by Lloyd was used for simple non-parametric integration with the sigmoid activation function. The study by Lloyd found that integration using neural networks in predictable and constant variability functions obtains reasonable outcomes. Integration using neural networks was not the best method compared to the classical numerical integration methods (which include the Adaptive quadrature method [2]), but it had a better result than some other methods (such as the trapezoidal rule). This proves that a neural network is useful for approximating the integrand function within a restricted integration region. When compared with splines [3], neural networks are more practical for approximating functions because they are universal approximators and generalise well to several dimensions. [4] On the other hand, traditional analytic integration approaches can scale to multiple dimensions, but analytic calculation complexity increases as the number of dimensions increases. [5],[6] The Monte-Carlo Integration (MCI) can be performed differently from the analytical approach in dimensional growth of integrand.[7] But the linear increase in computational time that can occur when there are many parameters can consume much time. However, the neural network integration has consistent precomputation time with less calculation time of the integrand rather than other methods. Therefore, previous research presents the usefulness of precomputation

and neural networks for the universal approximator. But, the research only focused on single non-parametrised result by comparing accuracy and computation time with differences in training data sizes. Consequently, these results only show the influence of training data size on neural network integration, which means the previous case study concentrated more on the methodology of neural network integration. Yet still, there are many influential factors to test the performance of neural network integration; thus, this research will focus on the expanded methodology and tests for practical use.

From the previous research, we inferred the insightful hypothesis that a neural network could be adapted to repetitive integration with multiple parameters to expand the usefulness of the case study. This proposition in this paper is specific to the hyper-cube in the Feynman integrals but could be generalised to other hyper-cube functions. In many areas, multi-dimensional parametric integration is required, and some peculiar equations have extremely hard to determine solutions. Thus, solutions that use numerical integration are beneficial and sometimes essential. Likewise, many complicated functions demand solutions by various parameters, but this parametric neural network integration research concentrated on the Feynman integrals as a sample integrand. There have been no substantial studies on parametric neural network integration for calculating the Feynman integrals. Therefore, this paper presumes that parametric integration in the sigmoid activation function can produce faster and more accurate results for the Feynman integrals. Also, the ReLU activation function expects to perform even better than the sigmoid activation function.
Thus, this research aims to expand from the existing concept of non-parametric integration to verify whether parametric integration provides reasonable accuracy and faster integration than the Monte Carlo method for the Feynman integrals.

This section introduces basic information about the new numerical integration methodology, Parametric Neural Network Integration (PNNI), to realise the hypothesis about fast parametric integration with high correct digits, which means the accuracy of the result. PNNI is a numerical integration algorithm with fixed parametric values that uses a single-layered neural network as an interpolation function. The main advantage of PNNI is that it can perform much computational processing in advance, allowing the neural network to be calculated quickly later. Network training is computationally expensive but only needs to be done once. When the training is finished, the network is designed to ensure that a parametric solution for integration is based on all network inputs. This enables the quick calculation of subsequent integration for various parameter sets.
The current study makes three major contributions as follows:

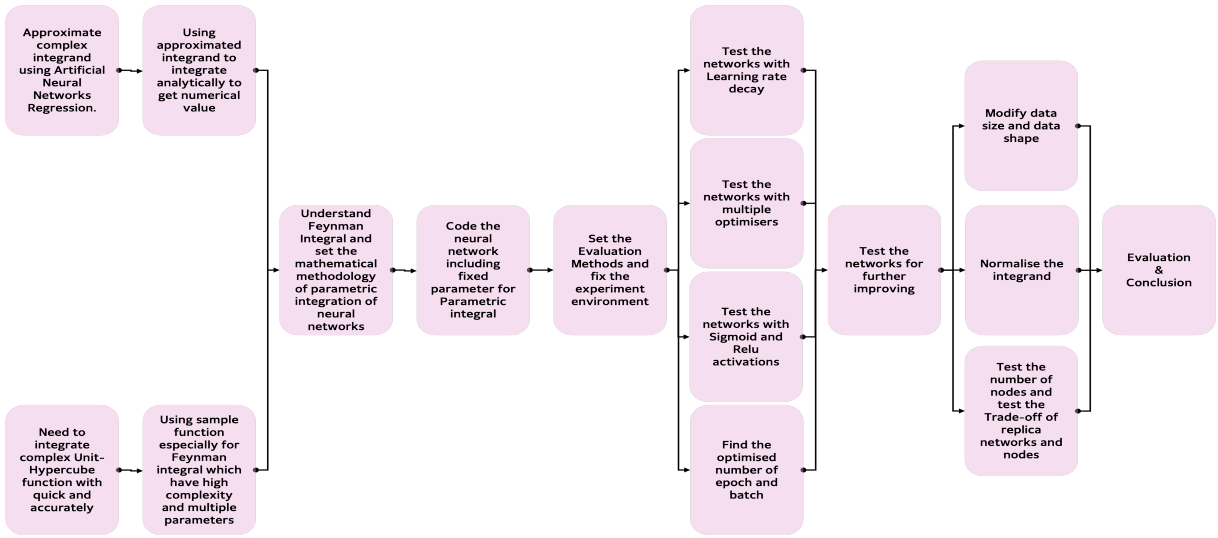


Figure 1: The architecture of progress in this paper.

1. Expand the neural network integration to PNNI.

2. Test performance with a sample integrand to determine PNNI could practically be used for the

Feynman integrals.

3. Provide insightful guidance to the community, discuss challenges, introduce issues, and identify future directions.

The architecture in Figure 1 shows the taxonomy of PNNI tests to be covered in this study in a hierarchically-structured way.

1.2 Basic of sample integrand for experiments

One sample integrand, a one-loop box Feynman diagram, has been used to test the performance of the PNNI. This Feynman integral shows the following form. This mathematical form comes from Feynman rules which convert the Lagrangian expression of the Feynman diagram.

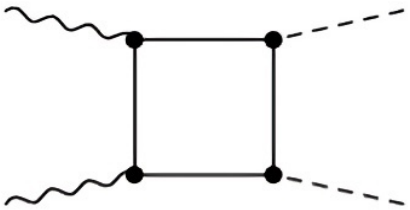


Figure 2: One loop of a box Feynman diagram. Left, gluons come and collide. Middle, make box loop by top quarks. Right. emit two Higgs bosons. Adopted Fig.3 from S.Nasuf[8]

$$I_{box} = \int \frac{d^D k}{i\pi^{D/2}} \frac{1}{(k^2 - m_t^2)((k+p_1)^2 - m_t^2)((k+p_1+p_2)^2 - m_t^2)((k-p_4)^2 - m_t^2)} dx \tag{1}$$

Equation 1 represents the integral of the one-box loop.[9] It shows two gluons collide, and they make a box of top quarks. These top quarks are coupled and emit two Higgs bosons. Where p_1, p_2, p_3, p_4 are satisfying momentum conservation in D dimensions, and m_t, m_h are the mass of top quarks and Higgs bosons. $\frac{d^D k}{i\pi^{D/2}}$ is dimensional regularisation (see Appendix A.1). To compute the result of this complex equation, Feynman parametrisation (see Appendix A.2) was used to simplify and applied sector decomposition (see Appendix A.5) to avoid singularities in the Euclidean plane. The evaluation of the PNNI was carried out as the third function of three sector-decomposed functions. This sample integrand can be used as an evaluation function for rigorous evaluation.

$$G = \int_0^1 \frac{1}{F_1^2} dx_1 + \int_0^1 \frac{1}{F_2^2} dx_2 + \int_0^1 \frac{2}{F_3^2} dx_3 \tag{2}$$

In Equation 2, a one-box loop Feynman integral is decomposed into three sectors. Initially, the one-box loop has four sectors, but the 3^{rd} and 4^{th} sectors have the same polynomial F_3, so the 3^{rd} and 4^{th} sectors were merged into one term of F_3 polynomial. Each term has its polynomial factor F_1, F_2, F_3, which has three integral variables and four property variables. Following Equation 3 is the 3^{rd} sector, which is formed with polynomial F_3.

$$\frac{2}{\begin{aligned}(m_t^2 + 2x_3 m_t^2 - x_3 m_h^2 + x_3^2 m_t^2 + 2x_2 m_t^2 - x_2 s_{12} + 2x_2 x_3 m_t^2 - x_2 x_3 m_h^2 + x_2^2 m_t^2 + 2x_1 m_t^2 \\ +2x_1 x + 3m_t^2 - x_1 x_3 s_{14} + 2x_1 x_2 m_t^2 + x_1^2 m_t^2)^2\end{aligned}} \tag{3}$$

x_1, x_2, x_3 = Integral parameters (integral variables)
$s_{12}, s_{14}, m_h^2, m_t^2$ = Input fixed parameters (property variables)

In Equation 3, integral variables are always constraints in the $[0, 1]$ range (see Appendix A.3 to Appendix A.5), and the input fixed parameters are constraints in $[-30, -3]$ to avoid the imaginary section of the integral value (restriction of the function in Euclidean region).

The one-loop box diagram looks to be simple, as shown in Figure 2, but the mathematical terms in Equation 3 are difficult to integrate analytically.

1.3 Artificial Neural Networks

For mathematically easy integration using weights and biases, artificial neural networks (ANNs) were used to approximate the sample integrand, Equation 3. The main concepts and mathematical representations that were used to build PNNI are as follows.

ANNs are statistical learning algorithms inspired by biological and cognitive neural networks (particularly the central nervous system). The artificial neural network is an overall paradigm in which artificial neurons (nodes) build networks the same way synapses are combined in the brain, feedback adjusting the bound output of nodes through learning, and hence ANN has problem-solving skills.

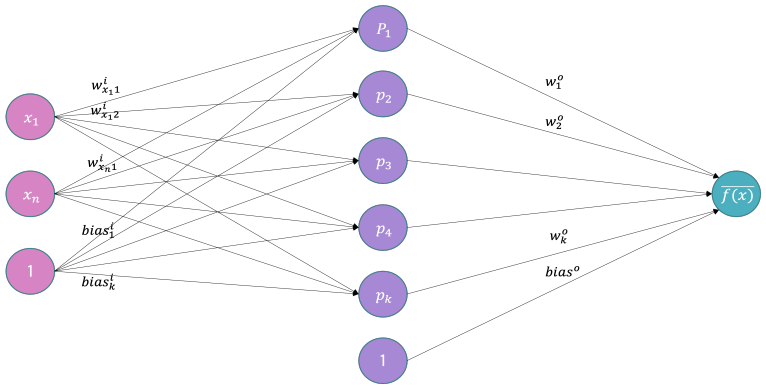


Figure 3: A simplified representation of an Artificial Neural Network with single layer of nodes(shallow neural network) that was used in the integration. This network train with k number of nodes and s number of dimensions.

The ANN in Figure 3 constructs Multiple Layer Perceptrons (MLPs) to approximate the nervous system. The perceptron (node), which corresponds to a neuron, accepts input, weights, and bias. The output of the node is the result of the activation function, which is determined by the sum of the weights multiplied by the input and adds bias. The PNNI is calculated using activation functions (sigmoid, ReLU), weight (input, output), and bias (input, output). Therefore, all node values follow the function of sigmoid and ReLU to obtain the actual value rather than using the threshold (0 or 1) of the node.

$$\overline{f}(x) = b^o + \sum_{j=1}^{k} w_j^o \, \phi(b_j^i + \sum_{s=1}^{n} w_{js}^i x_s) \tag{4}$$

The method of constructing an ANN(Figure 3) can express mathematically to Equation 4, where

- b^i and b^o represent biases input and output of single layer network respectively, which are the elements of $b^i \in R^a$, $b^o \in R$ (k is a number of nodes).
- w^i and w^o represent weight input and output of single layer network respectively, which are the elements of $w^i \in R^{a \times n}$,$w^o \in R^a$ (n is a number of dimensions (parameters)).
- Usually x_s is the elements of $x_s \in R^n$ but in this sample integrand, integral parameters have training data array in $x_s \in [0, 1]$ and fixed parameters (property variables) have training data array in $x_i \in [-30, -3]$ (s is total number of integral parrameters and i is total number of fixed parameters).
- ϕ is the activation function which can be Sigmoid(Equation 5) and ReLU(Equation 6), defined as

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

$$\phi(x) = max(0, x) \tag{6}$$

Owing to ANN approximation, calculating Equation 4 is easier to integrate than Equation 3.

2 Main Works and Basic Results

2.1 Parametric Integral of Neural Networks

This section defines the PNNI mathematically and solves each of the sigmoid and ReLU activation function cases in a tractable manner. Before integrating the approximated function, we define the rules that obtain a valid value of integration.

$$\left|\overline{f}(x_1 \dots x_s, c_1 \dots c_i) - f(x_1 \dots x_s, c_1 \dots c_i)\right| < \Delta \tag{7}$$

x_s : Integral parameter (integral variables)
c_i : fixed parameter (property variables)
s : Number of integral parameters (integral dimension)
i : Number of fixed parameters

1. In Equation 7(c_i is fixed parameters for parametric integral) any $\Delta > 0$ which can generate the real number of weight and bias. $w^i \neq 0$ is essential to avoiding singularity.

$$\overline{I}(x_1 \dots x_s, c_1 \dots c_i) = \int_{\alpha_1}^{\beta_1} \cdots \int_{\alpha_s}^{\beta_s} \overline{f}(x_1 \dots x_s, c_1 \dots c_i) dx_1 \dots dx_s \tag{8}$$

2. The integral of approximated function has a closed-form formulation which makes the result finite. Where α_s,$\beta_s \in R$ Equation 8 in every dimension. (In the sample integrand case, it finite the integral from $[0, 1]$)

3. The integrand function $f(x)$ is continuous

Each case of the sigmoid and ReLU activation integrals will be checked using these rules.

2.1.1 Parametric neural network integral with logistic Sigmoid activation

The sigmoid integration was generalised as a non-parametric integral in reference [1]. This section traces the details of parametric sigmoid integration.

One of the major activation functions, the logistic Sigmoid, is the same form as the 0^{th} order of the polylogarithm (or Jonquière's function). If $e^{-z(x)}$ ($z(x)$ which is a linear discretionary function) in polylogarithm, it follows a sigmoid form in Equation 9.

$$-Li_0(-e^{z(x)}) = \frac{1}{1 + e^{-z(x)}} \tag{9}$$

Equation 9 can continue to integrate the polylogarithm.

$$\int -Li_k(-e^{z(x)}) dx = -(\frac{\partial z(x)}{\partial x})^{-1} Li_{k+1}(-e^{z(x)}) \tag{10}$$

$\frac{\partial z(x)}{\partial x}$ is constant because of $z(x)$ in Equation 10 is linear in x.
Assume the $z(x_n)$ is linear in x_n ($n = 1, 2, \dots$); only in the 0^{th} polylogarithm Li_0 has a different form which,

$$\begin{aligned} &\int_{\alpha_n}^{\beta_n} \cdots \int_{\alpha_1}^{\beta_1} -Li_k(-e^{z(x_n)}) dx_1 \dots dx_n \quad , k = 0 \\ &= \int_{\alpha_n}^{\beta_n} \cdots \int_{\alpha_2}^{\beta_2} (\beta_1 - \alpha_1) - (\frac{\partial z(x_1)}{\partial x_1})^{-1} Li_{k+1}(-e^{z(x_n)}) dx_2 \dots dx_n \end{aligned} \tag{11}$$

After the 0^{th} polylogarithm, one constant term $(\beta_1 - \alpha_1)$ in Equation 11 survives until the nth integration ($k \in n$). This is the characteristic term in sigmoid integration.

The three-dimensional parametric integration in Equation 12 has two integral parameters and one fixed parameter. In this case, the function integrates two times only for integral parameters.

$$I(x_1, x_2, c_1) = \int_{\alpha_2}^{\beta_2} \int_{\alpha_1}^{\beta_1} f(x_1, x_2, c_1) dx_1 dx_2 \tag{12}$$

This example considers two integral parameters as the first and second dimensions and the third dimension as a fixed parameter. Thus, this example will consider three dimensions for the approximated integrand using the neural network.
Likewise, fitting the ANN into the Feynman integral sample integrand also includes fixed parameters. Therefore, three integral parameters and four fixed parameters are considered as seven dimensions of function.

$$\overline{I}(x_1, x_2, c_1) = \int_{\alpha_2}^{\beta_2} \int_{\alpha_1}^{\beta_1} b^o + \sum_{j=1}^{k} w_j^o \, \phi(b_j^i + c_1 w_{j3}^i + \sum_{s=1}^{2} w_{js}^i x_s) dx_1 dx_2 \tag{13}$$

k : Number of nodes
After training the ANN to create the approximated integrand Equation 13, the integration of one fixed parameters will be considered as a constant in the approximated integrand. Thus, the constant term can deal with any values as input after training the network to get the integral result.

The calculated integral value of the approximated integrand Equation 13 becomes Equation 14.

$$= b^o(\beta_1 - \alpha_1)(\beta_2 - \alpha_2) + \sum_{j=1}^{k} w_j^o \, [(\beta_1 - \alpha_1)(\beta_2 - \alpha_2) + \frac{\Phi_j}{w_{j1}^i w_{j2}^i}] dx_1 dx_2 \tag{14}$$

where,

$$\begin{aligned} \Phi_j = & \, Li_2[-exp(-b_j^i + c_1 w_{j3} - w_{j1}^i \alpha_1 - w_{j2}^i \alpha_2] \\ & - Li_2[-exp(-b_j^i + c_1 w_{j3}^i - w_{j1}^i \alpha_1 - w_{j2}^i \beta_2] \\ & - Li_2[-exp(-b_j^i + c_1 w_{j3}^i - w_{j1}^i \beta_1 - w_{j2}^i \alpha_2] \\ & + Li_2[-exp(-b_j^i + c_1 w_{j3}^i - w_{j1}^i \beta_1 - w_{j2}^i \beta_2] \end{aligned} \tag{15}$$

In the function, Φ_j(Equation 15) has a constant term in each polylogarithm part. The constant term, w_{j3}^i is the 3^{rd} input weight of the network, which approximates the three-dimensional integrand, and c_1 is the fixed parameter value (property value) from the input.
Expand Equation 15 with n-dimension, and it is generalised to Equation 16.

$$\Phi_j = \sum_{l=1}^{2^n} \varepsilon_l Li_n[-exp(-b_j^i + \sum_{a=1}^{t} c_a w_{j(n+a)} - \sum_{s=1}^{n} w_{js}^i \gamma_{s,r})] \tag{16}$$

n=number of integral parameters (integral variables)
t=number of fixed parameters (property variables)

$$\gamma_{s,r} = \left\{ \begin{array}{ll} \alpha_s, & \text{if } \lfloor \frac{r}{2^{n-s}} \rfloor \text{ is even} \\ \beta_s, & \text{else} \end{array} \right\} \tag{17}$$

$$\varepsilon_l = \prod_{d=1}^{n} (-1)^{\left\lfloor \frac{r}{2^{n-d}} \right\rfloor} \tag{18}$$

$\gamma_{s,r}$(Equation 17) is integration limit with s^{th} dimension and r^{th} summation element.
ε_l(Equation 18) is the sign of l^{th} sigmoid integration term of the node.
Integral of approximated fucntion process can be generalised with n-dimensional expansion.

$$\overline{I}(f(x_n, c_t)) = b^o \prod_{s=1}^{n} (\beta_s - \alpha_s) + \sum_{j=1}^{k} w_j^o [\prod_{s=1}^{n} (\beta_s - \alpha_s) + \frac{\Phi_j}{\prod_{s=1}^{n} w_{js}^i}] \tag{19}$$

Generally, this approach regards some fixed parameters (property variables) as constants in the integration process.

2.1.2 Parametric neural network integral with ReLU activation

Besides the sigmoid used in the case study, this section also researches using the ReLU activation function to achieve faster and more accurate calculation results.

The difference in the PNNI process between the logistic Sigmoid and ReLU is the integration of the activation function (ϕ).
In case of ReLU, we assume the $z(x_n)$ is linear in x_n ($n = 1, 2, \dots$), which is arbitrary argument.

$$\phi(z(x_n)) = max(0, z(x_n)) \tag{20}$$

$$\int_{\alpha}^{\beta} \phi(z(x_n))dx = \int_{\alpha}^{\beta} max(0, z(x_n))dx \tag{21}$$

$$\int_{\alpha_n}^{\beta_n} \cdots \int_{\alpha_1}^{\beta_1} \phi(z(x_n))dx \dots dx_n = \int_{\alpha_n}^{\beta_n} \cdots \int_{\alpha_1}^{\beta_1} max(0, z(x_n))dx \dots dx_n \tag{22}$$

$$\Phi_n = (\frac{\partial z(x_n)}{\partial x_n})^{-1} \frac{1}{(n+1)!} max(0, z(x_n))^{(n+1)} \tag{23}$$

$\frac{\partial z(x_n)}{\partial x_n}$ is constant because of the arbitrary argument $z(x_n)$ is linear in x_n.
From the example of section 2.1.1, the three-dimensional integrand has two integral parameters and one fixed parameter,

$$\Phi(z(x_2)) = \rho_1 \rho_2 \frac{1}{3!} max(0, z(x_2))^3 \tag{24}$$

where, ρ_1, ρ_2 are the constant values partial derivative of $z(x_1)$ and $z(x_2)$.
Assume $z(x) = x$, The n-dimensional integration of ReLU would be,

$$\Phi_n(x) = \frac{1}{(n+1)!} max(0, x)^{n+1} \tag{25}$$

This process can be generalised with n-dimensional expressions.

$$\overline{I}(f(x_n, c_t)) = b^o \prod_{s=1}^{n} (\beta_s - \alpha_s) + \sum_{j=1}^{k} w_j^o \frac{\Phi_j}{\prod_{s=1}^{n} w_{js}^i} \tag{26}$$

This method is simple because there is no polylogarithm in the analytic calculation process, so this is easier to generalise and faster to calculate.

2.2 Data for experiments

The data below were training data to be used in the experiments for this study, and the parametric properties and target values were collected as possible properties in the one-loop box Feynman diagram in the section 1.2. There were 27,000 parametric sets of data values. Therefore, a total of 27,000 individual integral values were obtained.

Usually, the Monte-Carlo Integration (MCI) method is used as the numerical integration method, and if it uses many training data points, it has a very small uncertainty. Therefore, in this study we use the Monte-Carlo Integration method as an original value that will be compared with PNNI values. From Equation 3, MCI obtain target values for evaluation criteria. From the characteristic of Feynman integral,

$s_{12}, s_{14}, m_h^2, m_t^2$, any one-fixed parameter could be 1.0 in the whole training data Table 1 (this training data uses one as the mass of a Higgs boson). Because fixed parameters depend on the ratio of property values, fixed parameters are divided by one of a property value. Therefore, a one-fixed parameter (one property variable in Table 1) can be set as 1.0 to simplify fixed parameters. If the other fixed parameters become a positive number, the integral may become much more complicated; this paper keeps the explanation simple. If the parameters have different signs, the polynomials in the

Monte-Carlo-Integral values						
N	$\mathbf{s_{12}}$	$\mathbf{s_{14}}$	$\mathbf{m_h^2}$	$\mathbf{m_t^2}$	$\mathbf{I(\bar{f})}$	ϵ (Uncertainty)
1	-3.01	-3.02	-3.03	1.0	3.42506... e-02	1.39692... e-15
⋮	⋮	⋮	⋮	⋮	⋮	⋮
30	-3.01	-3.02	-29.8	1.0	9.05274... e-03	3.90085... e-15
31	-3.01	-3.946...	-3.03	1.0	3.34308... e-02	1.40969... e-15
⋮	⋮	⋮	⋮	⋮	⋮	⋮
900	-3.01	-29.899...	-29.8	1.0	7.49880... e-02	3.88248... e-15
901	-3.94...	-3.02	-3.03	1.0	3.15801... e-02	1.45866... e-15
⋮	⋮	⋮	⋮	⋮	⋮	⋮
27000	-30.0	-29.89...	-29.80	1.0	2.73155... e-03	4.62784... e-15

Table 1: The result of the Monte-Carlo Integral $I(\overline{f})$ (target values) for different values of $s_{12}, s_{14},\ m_h^2,\ m_t^2$ and uncertatinties. Total number of data is 27,000.

sector decomposed function could be 0, which means the integrand could have singularities, and the x value (integral parameters) should go to the complex plane to detour the singularities. This study aims to determine the usefulness of PNNI in the Feynman integral. The sample integrand and training data are more simpler than other Feynman integrals for diagrams. However, the sample integrand is still more complex than many other hyper-cube functions.

The neural network integration was not tested using 27,000 separated parameters from the Feynman integral in other research, but in this experiment, 27,000 training data to measure the time to obtain whole target values and check the average accuracy of the network.

2.3 Evaluation Methods and Experimental Environment

This section describes the evaluation methods and fixed environmental settings used in the experiments. The hyper-parameters or methods that should be evaluated in the individual experiments were changed and used different factors.

"Correct Digits" means the number of digits that match target values. The target value of the correct digits were the MCI values. These 27,000 correct digits consist of the histogram, representing the average accuracy of PNNI.
Where,

$$-Log_{10}(\left|\frac{Target\ value(MCI\ value) - Estimated\ value}{Target\ value(MCI\ value)}\right|) \tag{27}$$

In Equation 27, the MCI value is the calculated value of the Monte-Carlo integral in Table 1. MCI value is highly reliable because it has a below 10^{-15} uncertainty value in Table 1. Therefore, the MCI value is appropriate to use as a comparison value of PNNI. A histogram of correct digits will show the performance of networks. If the correct digits trend of the histogram (median value of the histogram) is bigger, the network can estimate larger correct digits on average. All histograms are made using 27,000 fixed parameters in Table 1.

Most experiments use the same fixed environment. However, some experiments that need to change hyper-parameters were conducted after the change in hyper-parameter. All experiments were conducted in the environments below. The rationale for the environmental setup for the test was aimed to check the comparison results quickly. All the results of the experiments will be discussed in each section, and the best environmental setup will be summarised in the evaluation section.

- Neural Network Library: Keras (Tensorflow2)

- Training data size and shape: 50,789 Dataset with Quasi Monte-Carlo lattice and randomly shuffled training data array
- Activation function: ReLU Activation
- Optimiser: ADAM optimiser
- Number of Nodes: 125 Nodes
- Number of Networks: 1 network
- Number of Epoch: 100 Epochs
- Number of Batch: Default of Keras (Tensorflow2)
- Normalised Integrand selected

Because this evaluation method with a fixed environment obtained 27,000 correct digits and validated the median value as the PNNI performance, even a single histogram indicates 27,000 network verifications.

2.4 Basic Results by Hyper-Parameters of ANN

2.4.1 Learning Rates

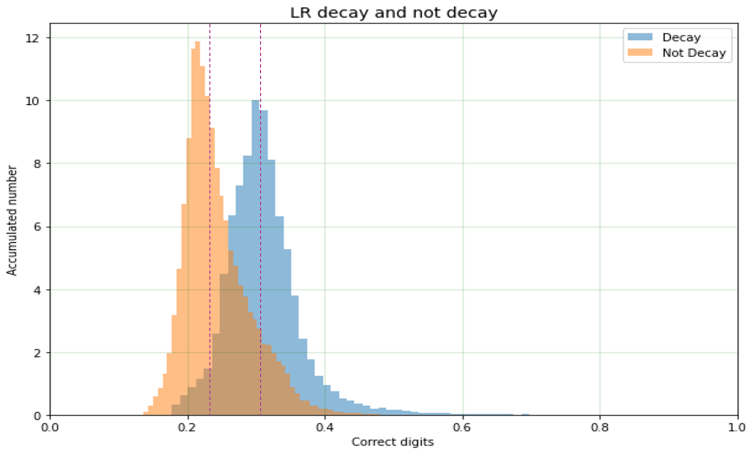


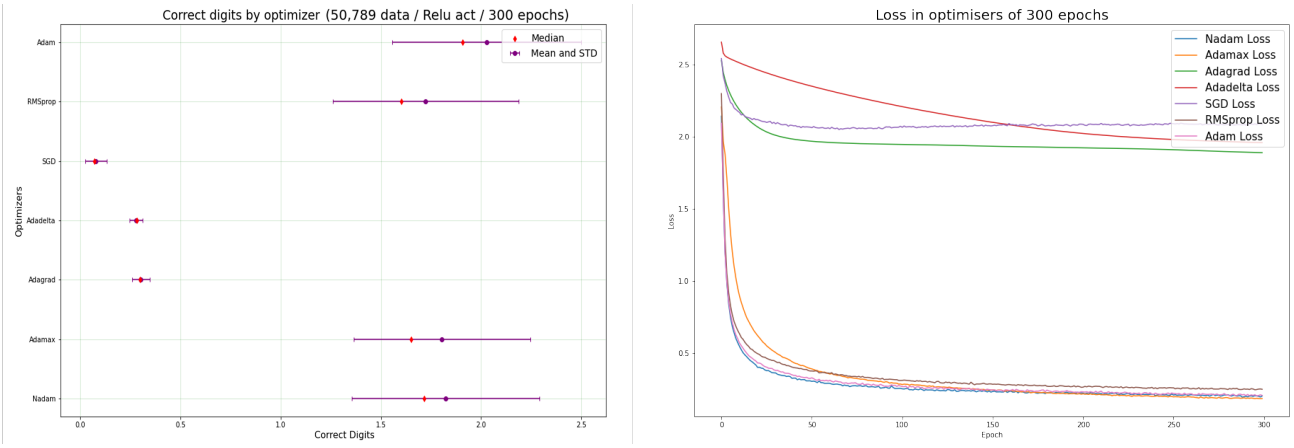
Figure 4: Correct digits compare between fixed learning rate and learning rate decay process. Because of SGD optimiser experiment could not converge until 100 epochs. So the difference in correct digits between the two methods has 0.08. However, it would be more extensive in the big number of epochs.

The learning rate determines the step size of neural network learning. This helps to find the convergence spot so that the neural network can approximate the integrand well. It is essential to set it up accordingly, and a technique called learning rate decay allows networks to reach faster and more accurate convergence points depending on the function. Therefore, this section verifies that the learning rate decay is used appropriately for PNNI.

This experiment was conducted using a fixed environment, but the first network was built with a fixed learning rate (0.01) SGD optimiser, and the second was built with a learning rate decay SGD optimiser. (In the case of the Adam optimiser, the learning rate is reduced in the algorithm itself, so it should not be used.) As a result in Figure 4, the network with learning rate decay showed 0.8 higher correct digits than the not decayed network and represented good results. (Since this experiment used SGD optimisers, the result only shows the effect of the presence or absence of learning rate decay rather than the absolute value of correct digits, so the experiment presents relative results.)

The high learning rate is a good strategy for avoiding the complicated loss surface for converging. Most real data sets have complexities in loss surface. Therefore, a high learning rate also needs to be used, but a high learning rate is difficult to find a small spot converging point. Accordingly, when it comes to the end of convergence, it is better to use a low learning rate. Above all, learning rate decay will help to overcome the difficulties in converging the loss.[10]

2.4.2 Optimisers (Solvers)



(a) (b)

Figure 5: (a) Correct digits measured from training data by optimisers. Adagrad, Adadelta and SGD does not have over one correct digit. (b) Loss of each network (Mean Absolute Error). Adam, Adamax, RMSprop and Nadam have similar losses but have different median correct digits in (a).

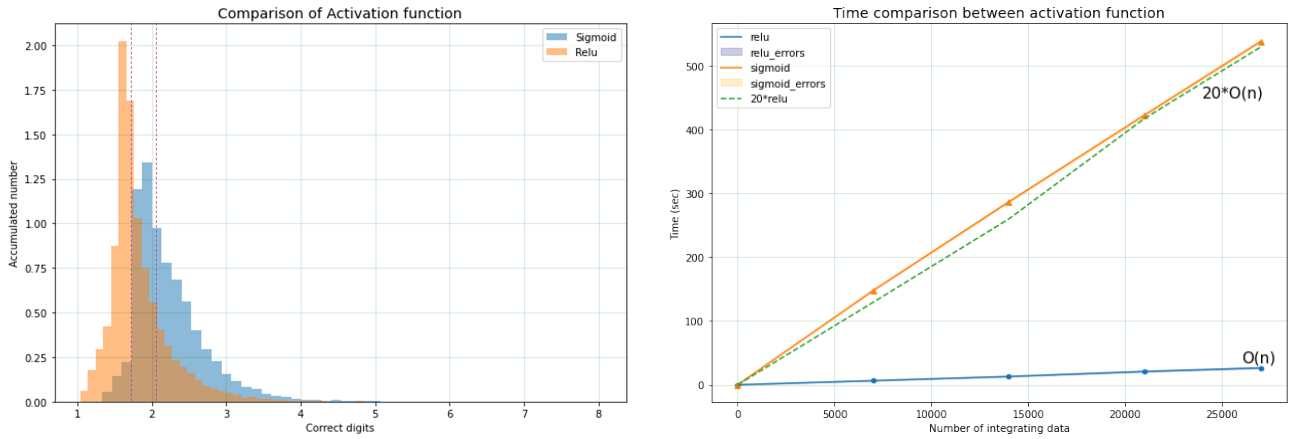
The optimiser is an algorithm that helps to find the optimal value by updating the differential value, which is feedback derived through back-propagation.
The ANN frameworks provide several optimisers (solvers), such as the Quasi-Newton method (limited memory-BFGS), stochastic gradient descent (SGD) method, and extended stochastic gradient descent called ADAM. Many optimisers have their own advantages for converging on different problems. However, in general, SGD has been used in many cases but is too slow, and L-BFGS is an excellent optimiser in low-dimensional situations. The sample integrand case has a high dimension, which is unsuitable for using L-BFGS.[11] Therefore, the Adam method is a lot faster and appropriate for high dimensions.[12] Unlike SGD, Adam is accelerated by force in the inclined direction. Adam moves on the loss surface the same as the ball moves as if it were rolling on the surface. Comparing SGD optimisation processes, the degree of the zigzag moving in the loss surface is less, and each parameter is adaptively adjusted to create a customised learning rate. Adam is a technique that combines momentum and adaptive AdaGrad optimisers. Adam sets three hyper-parameters: the learning rate, the primary momentum coefficient, and the secondary momentum coefficient. For example, if the coefficient for the primary momentum = 0.9 and the coefficient for the secondary momentum = 0.999 are set as default values, good results can usually be obtained.
Several optimisers are commonly used in ANN, including the RMSprop, AdaDelta, Adagrad, Adamax, Nadam, SGD, and Adam optimisers.

This experiment used the same sample integrand, the Keras (TensorFlow2) library, ReLU activation, and all other hyper-parameters in the same fixed environment. Each experiment used different Optimisers. The Adam optimiser optimised well for the sample integrand, Figure 5 (a),(b), shows the band of correct digits and value of loss of optimisers over the same epochs. The Adam optimiser was also one of the fastest optimisers in this experiment and got the highest (1.8 in median) correct digits with small losses. Hence, the Adam optimiser was a superb choice for the sample integrand. This experimental result came from the attributes of the Adam optimiser, which has a learning rate decay and momentum for fast convergence. However, SGD, AdaGrad, and AdaDelta optimisers have

relatively small correct digits in Figure 5 (a). Because of improper convergence methods for only in this sample integrand, they were not converged.

Optimisers determine the convergence method and find the convergence point. Although the performance of the Adam optimiser was good in the sample, it has the probability of missing the convergence zone depending on the function. Therefore, an appropriate solver must be found by measuring training loss to find the appropriate convergence point.

2.4.3 Activation function



(a) (b)

Figure 6: (a) Histogram of correct digits for both sigmoid and ReLU activation functions. sigmoid has 0.02 higher correct digits than ReLU. (b) Time comparison in each number of target values. sigmoid was 20 times slower than ReLU in computing. In the plot, the error means only computing time error, not the target values error.

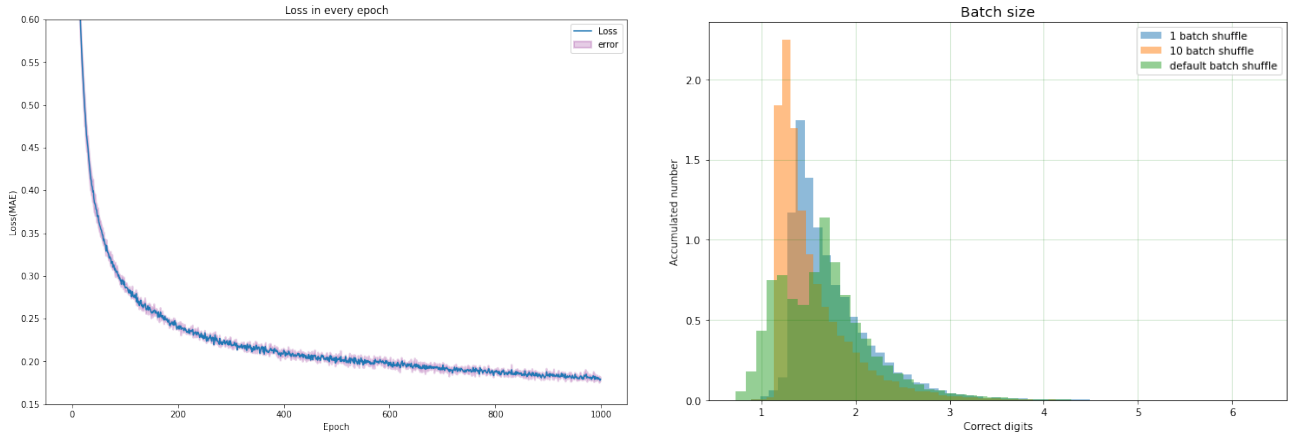
The activation function is an essential factor that determines the process of analytic integration in networks and influences the quality of the approximated function. It also has a significant influence on the integration speed of PNNI.

This experiment implemented the same sample integrand, Keras (TensorFlow2) library, and fixed environment. However, a different investigative element was the usage of the sigmoid activation and ReLU activation. Consequently, sigmoid activation showed 0.2 higher digits than ReLU activation in Figure 6 (a). The result in Figure 6 (a) shows that for this sample integrand, sigmoid activation provides better prediction results than ReLU activation, but for typical situations, the result is reversed. This phenomenon occurred because the sample integrand is differentiable, and sigmoid activation is also a differentiable function, but the ReLU activation is not. Moreover, Unlike other ordinary samples, the case of the sample integrand provides sufficiently predictable data and less noise approximating the function with neural network only does interpolation within the training data. For this reason, sigmoid activation gives higher correct digits than ReLU activation. If the neural network needs to extrapolate for model learning to evaluate outside of the range and ReLU activation does not bound within the same range (sample integrand bounded in $[0, 1]$), the result is expected to be better for ReLU activation.
Each calculation time was measured and expressed in Big O notation in Figure 6 (b). According to the test results, the tendency of calculation speed on ReLU activation follows O(n), and sigmoid activation also follows O(n). However, in the case of sigmoid activation, the calculation time was about 20 times slower than for ReLU activation. Because of sigmoid activation during the integration process, the computational complexity increased due to the polylogarithm operation, which is shown in Equation 16. This computation time was slower than the simple computation of ReLU activation,

which is shown in Equation 25.

As shown in the result figures above, the activation function determines the key elements of the usability of numerical integrals by artificial neural networks, and the activation function requires careful selection for each integrand. For the Feynman integral, the ReLU activation was suitable for quickly checking the performance and trend of PNNI in the tests of this research.

2.4.4 Number of epochs and batches



(a) (b)

Figure 7: (a) Loss graph in 1,000 epoch. Adam solver finds convergence quickly. The loss does not much difference after 200 epoch (b) Correct digits comparison with different numbers of batches in shuffled training data on QMC lattice. The number of batch was not much different under the same shuffled (decoupling. See 3.1.2) training data array.

The number of epochs and batches affects the training results of the PNNI. This fact led us to determine how much the number of epochs and batches could affect the results and what the best number was.

This experiment used a fixed environment and prolonged the number of epochs to determine the converging of loss in Figure 7(a). The loss does not dramatically change after 200 epochs but decreases slightly up to 1,000 epochs. An epoch is one cycle of training the neural network with all the training data. During an epoch, the network utilises all of the training data once. A forward and backward pass combined counts as one pass. An epoch comprises one or more batches, which is a portion of the training data for training the neural network. As the number of epochs increases, the loss decreases, but if there is no further change in back-propagation, the loss remains at a similar level. Therefore, this paper used appropriate numbers of epochs (about 200 epochs) to identify performance trends within a short time. However, experiments that require definite convergence were conveyed over 300 epochs. Using more than 300 epochs in practical uses is expected to yield better results. Though, the network should apply the appropriate number of epochs because it can lead to overfitting with too large epochs. In Figure 7(b), the correct digits were confirmed by changing the number of batches in a fixed environment. The number of batches did not significantly affect the correct digits because the sample integrand had many dimensions and parameters. Moreover, the training data used to approximate the function were randomly generated (the Quasi Monte-Carlo lattice was also randomly shuffled its order of the training data array), which also affected this results. Therefore, other experiments were conducted using the default number of batches because the default number of batches was an adequately optimised batch value. However, it is recommended that the number of batches be moderately subdivided. The appropriate batch size is required when other samples are used.

The number of epochs did not have a decisive impact on PNNI performance. But, about 300 epochs would be enough for other tests because many epochs lower the loss until it converges and increase computing time. Therefore, before using the PNNI, one considers the total computing time to set the appropriate size (at least 100 epochs are recommended).

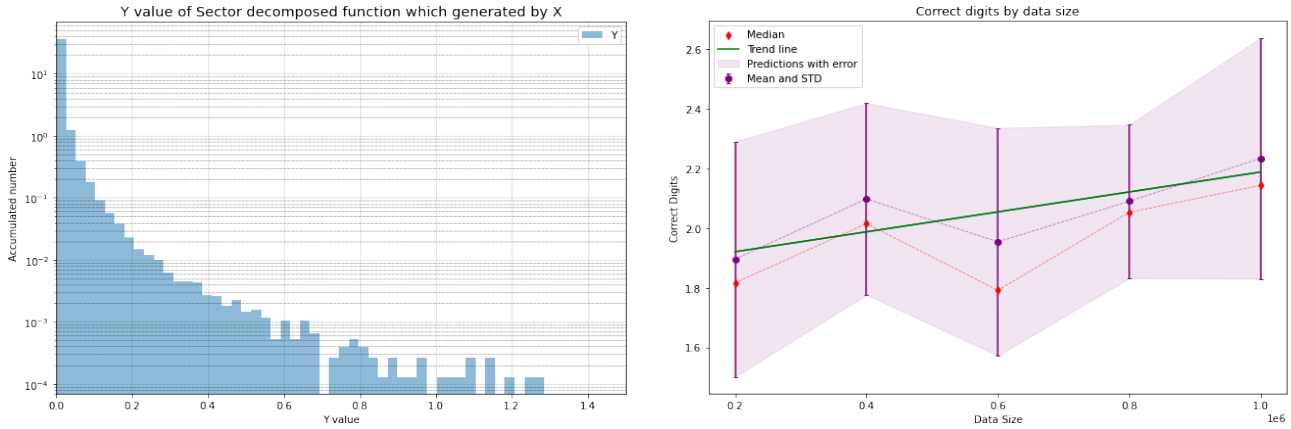
3 Further Experiments

3.1 Shape and Size of Data

For PNNI, training data can be generated from an existing integrand. This allows the PNNI to determine approximation quality from the shape and size of the training data. Thus, it is necessary to study the size and shape of training data that enables PNNI to perform better.

3.1.1 Size of data

The large training data size gets the information of local variance in the integrand to help approximate the function correctly.



(a) (b)

Figure 8: (a) N (300,000 in this figure) size training data of Y histogram, which used randomly generated X values to sample integrand. (b) Correct digits compare from 200,000 training data to 1,000,000 training data. Progression of correct digits slightly increased.

Training data generated by the random generator tool were used as the random lattice of variables x_n, c_t in Equation 19 integrand to train PNNI. Usually, large training data sizes have more accurately predicted target values because the data points provide more spatial information. Figure 8 (b) shows the inclination to increase correct digits with bigger training data sizes, but there was not much difference between relatively large and small data sizes. In addition, the 400,000 training data size had the high correct digits in Figure 8(b) due to the coincidentally well-applied weight and bias during the training of the network. Larger data sizes can reliably represent higher correct digits, but for efficiency (correct digits per computing time) in training, size can be selected at approximately 50,000 training data per dimension (for example, 300,000 training data in terms of six-dimensional integrand). (The training data size setting of 50,000 per dimension is roughly anticipated with empirical results. Furthermore, the sample integrand has a small range $[0, 1]$ in integral parameters. So the sample integrand could appropriately accept a relatively small training data size than other hyper-cube integrands, which have bigger integral ranges)

As the number of dimensions increases, the training data size should increase accordingly, which helps in attaining accurate results. For example, if one dimension has 100 uniformly distributed data points in $[1, 100]$, the length between each data point is 1, but if the dimension increases to two or

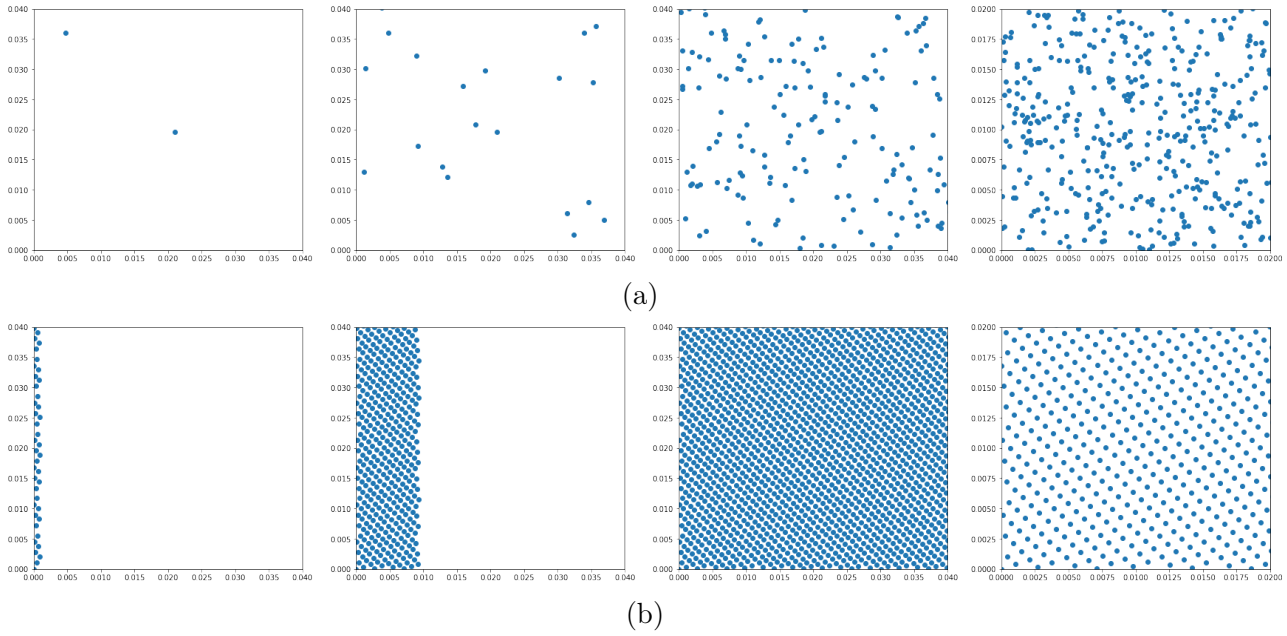


Figure 9: (a) Data generated by time flow in random training data generation function. Some areas were dense, but some were empty (b) Data generated by time flow in Quasi Monte-Carlo training data generate function. All areas have the same density.

three dimensions, the distance between data points is $\sqrt{2}$ and $\sqrt{3}$. Therefore, having a larger data size reduces the distance between the data points, which can provide information in empty spaces. Computing time also increases with training data size. Thus, it is crucial to consider the adequate training data size to save computer resources and time.

3.1.2 Shape of data

When the training data are generated using the sample integrand, the generating tool establishes how the training data spread in the hyper-spaces. This section compares the performance and efficiency of PNNI when it uses the random tool and the Quasi Monte-Carlo tool.

Figure 9 shows the random data lattice and the Quasi Monte-Carlo (QMC) [13] data lattice taking up hyper-space over time. The QMC lattice tool produces rank-one lattices with a prime number of points.

$$\{t\} = t - \lfloor t \rfloor \tag{28}$$

$$\delta_i = \{\frac{i\nu}{n}\}, \quad i = 0, 1, \ldots, n-1, \tag{29}$$

Where $\nu \in Z^s$ is a generating vector [14] for fast component-by-component construction; ν has no common factor with n, which is an s-dimensional integer vector δ_i which applied bracket method in Equation 28.[15] The *generating vectors* were constructed using an algorithm that works very quickly if the lattice size is prime number. This strategy can be used for Feynman integrals that are far above complicated than the sample integrand.
The QMC training data array of integral and fixed parameters randomly shuffled its order from the original QMC training data array because it affected learning on the artificial neural network. This shuffling was a process that decouples the correlation of integral and fixed parameters when generated. (Only if the QMC tool generates the same data size of integral and fixed parameters. Because the QMC tool generates integral and fixed parameters in the same ways.) Figure 9(a) shows some locations where the training data points were dense; some areas were empty. This unbalanced training data lattice presents the possibilities of overfitting or underfitting when the networks are training. The QMC training data generator indicates that the QMC lattice is uniformly distanced in Figure 9(b),

and it is expected to deal with the variance in hyper-spaces in the sample integrand. [16] This investigation clarified that the training data points should be filled uniformly and minimise the empty data spaces. The training data generator should generate a uniform filling of the data points to overcome the data holes. Therefore, the QMC training data generator did qualify for reducing local variances and implementing homoscedasticity for approximated functions.

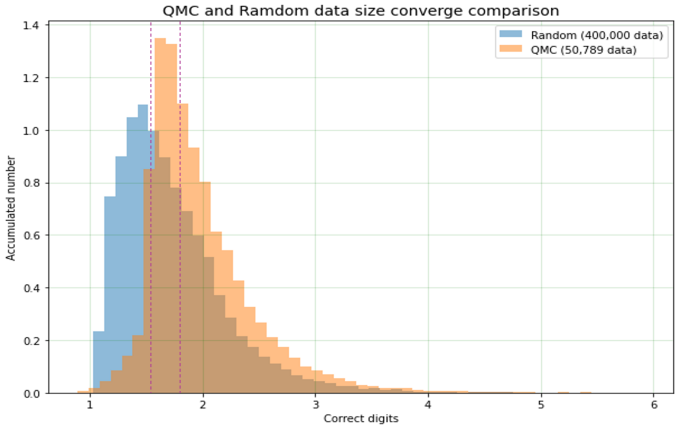
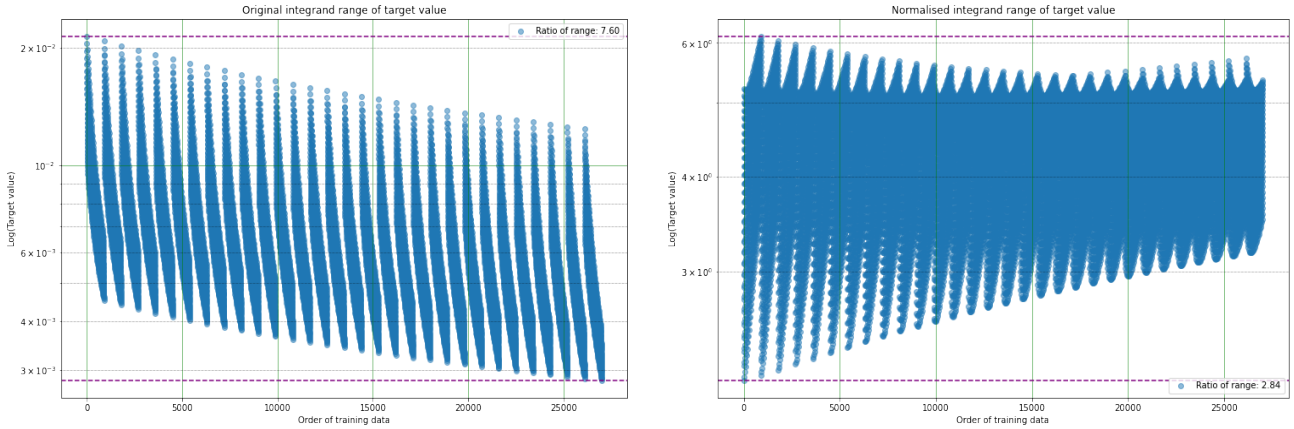


Figure 10: Correct digits of random training data lattice and QMC training data lattice, two lattices have different data sizes. Small training data size with QMC lattice has better (0.3) correct digits.

In Figure 10, the correct digits of the QMC lattice are 0.23 higher than the random lattice, which has nearly 7.5 times more training data points than the QMC lattice. Because of the uniformity of the training data points, the QMC lattice distributes training data points evenly across most locations in the integrand, effectively preventing the training data points from overfitting or underfitting in certain areas and also reducing the local variance. Thus, a QMC lattice with a small size can perform better than a big random data lattice. In particular, the training data size was reduced, which significantly reduced the time spent on learning for PNNI.

As a result, the size and shape of the training data affect accuracy, processing time, and overall PNNI stability. Therefore, training data plays a major role in the advanced PNNI.



(a) (b)

Figure 11: (a) Range of sample integrand within the various fixed parameters data set in Table 1. The ratio of minimum and maximum was 7.60. (b) The range of normalised integrand within the various fixed parameters data in Table 1. The ratio of minimum and maximum was 2.84, which has a smaller range than the original integrand.

3.2 Normalised Integrand

When PNNI approximates the original integrand, it is not easy to approximate using the neural network if the ratio of the integrand range is big. Therefore, if it is possible to reduce the ratio of the integrand range, the PNNI is approximated more closely to obtain accurate results. Accordingly, this experiment was conducted using the method of the normalised integrand.

The range of integrand values can span a wide scale for different values of the parameters $c_1, \ldots, c_t$, in Figure 11(a). And this method is useful to normalise the integrand by its value at a fixed location in x space. In practice, this entails conforming to a modified integrand,

$$\widehat{f}(x_1, \ldots, x_n, c_1, \ldots, c_t) \equiv \frac{f(x_1, \ldots, x_n, c_1, \ldots, c_t)}{f(\frac{1}{2}, \ldots, \frac{1}{2}, c_1, \ldots, c_t)} \tag{30}$$

and the estimate of the integral is normalised.

$$\widehat{I}(x_1, \ldots, x_n, c_1, \ldots, c_t) \equiv \frac{I(c_1, \ldots, c_t)}{f(\frac{1}{2}, \ldots, \frac{1}{2}, c_1, \ldots, c_t)} \tag{31}$$

Three input integral values and four fixed values in the integrand Equation 3 were used as a sample to create an integrand value. The wide range of integrand values may occur poor approximation to the neural networks. Therefore, we normalise the integrand by its value at a fixed location in integral hyper-spaces. This experiment chose the centre of the unit hyper-cube as a fixed location in Figure 11(b).

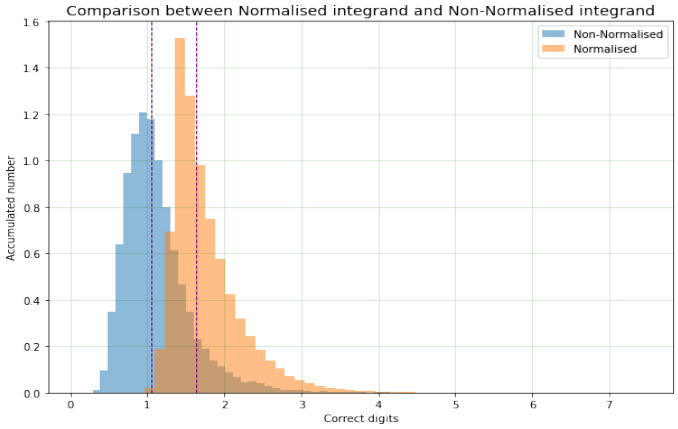


Figure 12: Compare correct digits between normalised integrand and original integrand. Normalised integrand has higher correct digits.

The experiment in Figure 12 measured the training results for the normalised integrand and the original integrand in a fixed environment. The results clearly showed that the results of the normalised integrand have 0.56 higher digits than the original integrand.

The normalised integrand is an excellent way to avoid the problem of an incomplete approximated function. There are many ways to normalise a hyper-cube function. For example, $x_n = \frac{1}{4}$ could also be the normalising factor in integrand hyper-spaces, or different normalising factors could be set each time. However, the centre of the hyper-cube was the most efficient (the range ratio of 2.84 was the smallest ratio) and convenient to apply analytic integration of the neural network.

3.3 Number of Nodes

3.3.1 Number of Nodes in one network

It is crucial to determine the ideal number of nodes in neural networks because the wrong number of nodes can cause the problem of underfitting and overfitting.

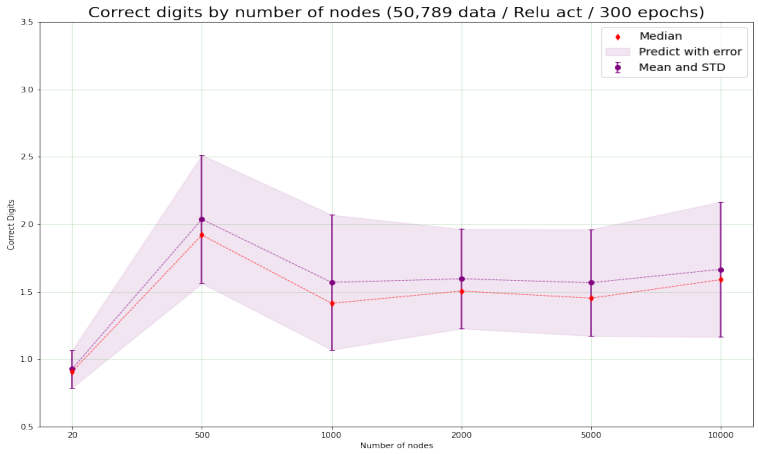


Figure 13: The number of nodes in single network. The correct digit was highest in 500 nodes, a hundredth of training data size.

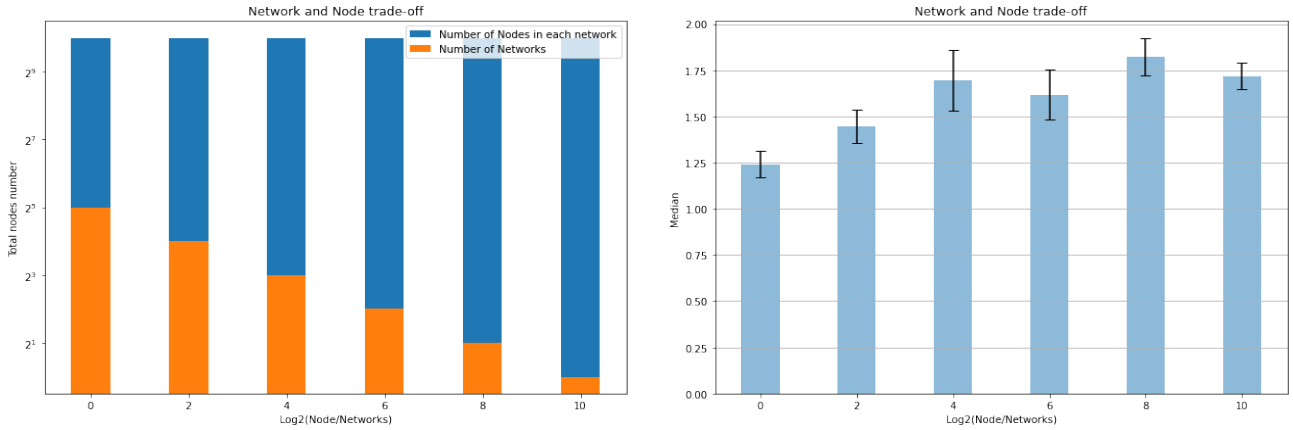
Figure 13 represents the correct digits by the number of nodes that can be held on a single network in a fixed experimental environment. In the experiment, we verified correct digits by increasing the number of nodes from very small to large numbers. A large number of nodes is about a fifth of the network training data size. The result shows that a small number of nodes may not give a good approximate shape for the sample integrand, and a relatively large number of nodes cannot predict well either. This implies that a small number of nodes occurred under-fitted because a few nodes cannot represent much of the complexity in the original integrand. Conversely, a large number of nodes were over-fitted because many of the nodes were affected by the back-propagation of relatively small training data points. So that approximated integrand from the neural network has high variance. Therefore, in the network with the current sample integrand, the highest correct digits was 500 nodes (in 50,789 training data size, ReLU activation function, and 300 epochs), which is nearly 100 times smaller than the size of the training dataset. Baum and Haussler's research [17] demonstrates that the number of weights should be at least ten times smaller than the sample size. Thus, the decomposed Feynman integral function has the best result in a range that is nearly 100 times smaller than the training data size.

The training data size and integrand complexity resolve the number of nodes. Thus, there is no formula or exact solution [18] for the number of nodes. It is essential to check the proper number of nodes in the PNNI for every integrand.

3.3.2 N number of replicas size trade-off

The network has different weights and biases each time it is trained, which increases the uncertainty of the integral results. Sometimes the network does not obtain a satisfactory integral result. Therefore, it is beneficial for multiple replica networks with one training data to compute the average value of integral results to stabilise and reduce uncertainty. This concept comes from the Law of Large Numbers. Getting the average value of replica networks clearly provide good accuracy, but it increases the time needed to train multiple networks. However, a new question arises as to what happens if the sum of all network nodes fix to 2^{10} and changes the number of replica networks and nodes in each replica. Accordingly, this experiment clarifies the effectiveness of the trade-off concept for nodes.

The effect of the replicas size trade-off is shown in Figure 14 (b). Increasing the number of replica networks while reducing nodes in a single network indicates a relatively insufficient result. The smaller the number of replica networks, the better the results. However, as the number of nodes in a single network gets closer to 2^{10} nodes, the less impact in correct digits again. Consequently, the number of replica networks that achieves the best results is within the range of $2 \leq number\ of\ networks \leq 16$. Therefore, it is evident that more than one individual network can provide more stable and better results, but too many networks cannot.



(a) (b)

Figure 14: (a) Sum of nodes in each replica and number of replica networks which was 2^{10} (b) Size trade-off of replica networks. The ratio of Node and Networks 8 was the best result.

Results show that a relatively small ratio of nodes and replica networks tends to have under-fitted, and a relatively big ratio tends to have over-fitted. Therefore, the number of nodes in a single network affects accuracy more than the number of replica networks. Moreover, ratio 8 in Figure 14 (b), has the best number of nodes 2^9 (Have a similar result to section 3.3.1) in each network, has the highest correct digits in the trade-off experiment. Yet, numerous replica networks still have a positive effect on accuracy. Even though ratio 4 has slightly under-fitted in replica networks, the average of correct digits is as much as ratio 8 result.

4 Evaluation and Conclusion

4.1 Evaluation

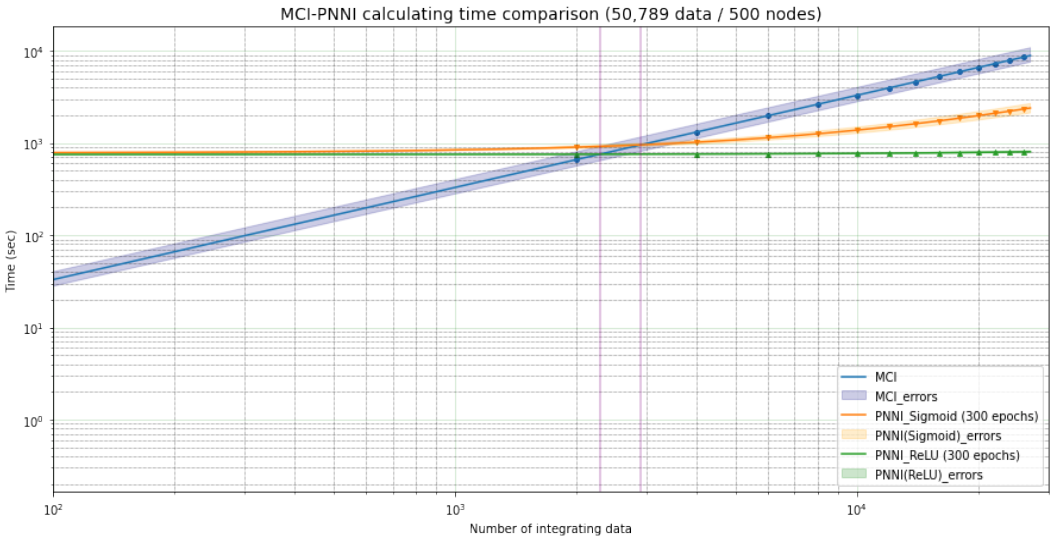


Figure 15: Comparison of computing time in multiple target values by numerical integration tools which use sample Feynman integral based on 500,000 data size. PNNI(Relu) have advantage after 2700 multiple target values and PNNI(Sigmoid) have advantage after 3000 multiple target values. In the plot, the error means only computing time error, not the target values error.

The overall performance of PNNI can be evaluated by checking the computational time in each tool and comparing the correct digits of all implemented methods.

In Figure 15, PNNI shows a relatively fast computational speed on the same training data size as MCI, except for the initial preparation time for network learning. However, MCI takes almost the same time for all individual results, which is greatly affected by the training data size. Moreover, extensive training data can yield individual results with lower uncertainty, but large training data takes a long time to compute for single results and linearly increase time for many results in MCI. PNNI, on the other hand, has a great advantage when it needs to calculate an enormous number of target values. A larger training data size can produce more accurate results for PNNI, but it is not a significant factor that changes accuracy as much as MCI. Previous experiments have shown that sigmoid activation has a calculation speed about 20 times slower than ReLU activation. However, the gap between the two methods in Figure 15 is significantly smaller than the gap between MCI calculation time and ReLU calculation time. Therefore, the use of PNNI is advantageous and has excellent potential if there are many target values, regardless of which activation function is used.

Median of Correct digits table in 27,000 Results				Relu Activation function (Keras)								
				Correct digits			Computing Time (Sec)			Efficiency		
				Data size(QMC grid)			Data size(QMC grid)			Correct digits/Time(min)*100		
				50,000	100,000	300,000	50,000	100,000	300,000	50,000	100,000	300,000
Network size	2	Node size	150	1.2	1.6	2.0	725.2	1519.2	4266.1	9.66	6.30	2.76
			500	1.8	1.8	1.7	749.0	1530.6	4415.4	14.14	7.06	2.37
			2000	1.6	1.8	2.3	744.7	1573.3	4410.2	12.97	6.99	3.06
			5000	1.7	1.6	1.9	889.5	1711.8	4522.6	11.39	5.74	2.46
			10,000	1.6	1.8	1.9	985.8	1803.5	4600.5	9.50	5.89	2.48
			30,000	1.8	1.3	1.7	1403.8	2208.8	5119.3	7.61	3.51	1.95
	4	Node size	150	1.5	1.4	2.0	1588.1	3364.4	9759.0	5.82	2.56	1.21
			500	1.8	1.6	1.9	1631.2	3487.3	9277.7	6.71	2.73	1.20
			2000	1.7	1.8	2.0	1629.9	3474.6	9368.8	6.37	3.17	1.26
			5000	1.5	1.8	1.6	1936.5	3715.3	9875.6	4.63	2.86	0.97
			10,000	1.8	1.9	2.0	2131.0	3982.6	10202.3	5.12	2.87	1.16
			30,000	1.8	1.6	2.0	2964.4	4707.4	10595.0	3.56	2.09	1.15

Table 2: Table of correct digits and computation time with several variations of PNNI methods. The number of nodes in the table is the number in each network. Higher correct digits, computing time and efficiency were illustrated in a darker colour. In the case of efficiency, not all high-efficiency results guarantee the best output. It is only for reference.

Table 2, which lists various methods, provides the accuracy of PNNI. First, the table shows that the correct digits of PNNI increase with the data size. However, the impact is minimal. In addition, as the data size grows, the calculation time multiplies. Furthermore, the number of nodes in the network is positively correlated to the data size. (see Appendix A.6) The number of nodes around one-hundredth of the training data size in the network shows the highest correct digits in the same training data size in Table 2. The number of networks also affects accuracy, but as with the data size, it does not impact accuracy significantly. The results show that using about two to three replica networks is efficient. The table was created to get the correct digits of the sample integrand using PNNI of the Adam optimiser and ReLU activation. If sigmoid activation had been used, the correct digits would be higher than if ReLU activation had been used, but the calculation time would increase significantly due to the polylogarithm calculation. Furthermore, the reproduction of the result table could have slightly different results because obtaining weights and biases of the neural network change every time, even though using the same training data.

Previous experiments have implemented various hyper-parameters in PNNI and diverse methods that could improve the accuracy of networks. The results of PNNI changed in several situations. Moreover, the hypothesis assumed in the introduction section, 'parametric integration provides affordable accuracy and faster integration than the Monte-Carlo method in the Feynman integral', was not completely achieved in the accuracy part from the correct digits results of each experiment. A summary of experiments for obtaining the best integral value using PNNI follows.

- Data size and shape: over 50,000 data per dimension with Quasi Monte-Carlo lattice
- Activation function: Sigmoid Activation function
- Optimiser: ADAM optimiser
- Number of Nodes: Around $\frac{1}{100}$ of data size
- Number of Networks: 2 to 4 Networks
- Number of Epoch: Over 300 epochs
- Normalised Integrand

4.2 Conclusion

We extended the methodology from the neural network integration shown in the previous case study. This cutting-edge technique, PNNI, has been specified to get multiple results in a short time from various parameter sets. Validation has been focused on evaluating accuracy and computation time through several experiments, especially in the decomposed Feynman integrals in the Euclidean domain. Nevertheless, PNNI could expand usability to other n-dimensional functions which need to integrate parametrically with multiple results.

The test methods were proposed for accuracy and evaluation time using many hyper-parameters for basic works (such as learning rate decay, the activation function and optimisers) and further works (such as the shape of the data grid, normalised integrand, and trade-off of network replicas) to improve the PNNI result and compare it with that of MCI. As expected at first, it was verified that the parametric integration of neural networks using the ReLU activation function progressed 20 times quicker. However, it was confirmed that the correct digits were 0.02 lower than the correct digits of the sigmoid activation function for the Feynman integrals. In the overall performance of PNNI, the integration using 27,000 individual sets of parameters generally shows two-digit accuracy. This result is low in accuracy for practically using PNNI for the Feynman integrals calculation, which usually needs six digits accuracy. But, if improvements are made in accuracy, PNNI could be a more powerful parametric integral calculator than MCI. Although the accuracy of the experimental results was poorer than expected, the potential can be found in the derived results. The current sample integrand can be seen as a high complexity function, and the complexity is mainly related to the number of dimensions and the variance in the integrand. Therefore, PNNI can provide higher correct digits for functions having little complexity, such as the oscillatory function.

Finally, PNNI was demonstrated to be a viable integration method for the methodologies presented in this paper. Moreover, PNNI is a good choice for multiple parametric integrations requiring less than three correct digits. The findings of this study suggest that more research into the application of training data forms can improve accuracy and computing time. Furthermore, training data normalisation has the potential for fast computational time and accurate approximation. Above all, the PNNI used in the experiments was formed with a single layer. However, it is expected to produce more accurate results if PNNI increases the layer depth to two or three layers. In fact, if the networks have the same number of nodes but have a different number of layers, for example, 500 nodes in single-layer networks and 50 nodes in 10-layer networks, the latter network shows better prediction results (see Appendix A.7 about the influence of layer-node trade-off). Therefore, studying deeper neural network integration will play a major role in improving the application of PNNI.

A Brief background of Feynman parametrisation to sector decomposition

A.1 Dimensional Regularization for Divergence of Loop integral

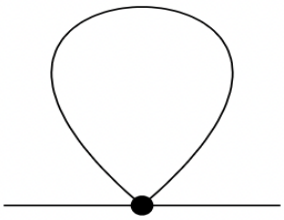


Figure 16: Tadpole Feynman diagram

Loop integral will be divergent if $k^2 -> \infty$, which is called an ultraviolet (UV) divergence. If $k^2 -> 0$, which is called an infrared(IR) divergence. If we consider tadpole integral in 4-dimensional spaces, it approximate to mathematical way in pola-coordinate.

$$\begin{aligned} &\int_{\infty}^{\infty} \frac{d^4k}{(2\pi)^4} \frac{1}{(k^2 - m^2 + i\delta)} \\ &= \frac{-i}{(2\pi)^4} \int d\Omega_3 \int_0^{\infty} dr r^3 \frac{1}{(r^2 + m^2 + i\delta)} \end{aligned} \tag{32}$$

$$\begin{aligned} &= \frac{-i}{(2\pi)^4} \frac{2\pi^2}{\Gamma(2)} \int_0^{\infty} dr r^3 \frac{1}{(r^2 + m^2 + i\delta)} \\ &\frac{-i}{(2\pi)^4} \frac{2\pi^2}{\Gamma(2)} \frac{1}{2} [r^2 - m^2 ln(r^2 + m^2]_0^{\infty} \end{aligned} \tag{33}$$

In Equation 33, r^3 makes the equation divergent. Therefore, treat number of space-time dimensions ($D = 4 - 2\epsilon$) from the current perturbation theory.
For this, we need to know the properties that is
(1) Linearity : For any complex numbers a and b.

$$\int d^d p[af(p) + bg(p)] = a \int d^d p f(p) + b \int d^d p g(p)$$

(2) Scaling : For any number of s.

$$\int d^d p f(sp) = s^{-} d \int d^d p f(p)$$

(3) Translation invariance : For any vector q.

$$\int d^d p f(p + q) = \int d^d p f(p)$$

Using the regularization properties in Equation 33 4-dimension could replace to D.

$$\begin{aligned}
&\int_{\infty}^{\infty} \frac{d^D k}{(2\pi)^D} \frac{1}{(k^2 - m^2 + i\delta)} \\
&= \frac{-i}{(2\pi)^D} \int d\Omega_{D-1} \int_0^{\infty} dr r^{D-1} \frac{1}{(r^2 + m^2 + i\delta)} \\
&= \frac{-i}{(2\pi)^D} \frac{2\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2})}) \int_0^{\infty} dr r^{D-1} \frac{1}{(r^2 + m^2 + i\delta)}
\end{aligned} \tag{34}$$

We can substitute r^2 to $y(m^2 - i\delta)$ to Equation 34 by the Scaling propertiy.

$$\begin{aligned}
I &= \frac{-i}{(2\pi)^D} \frac{2\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2})} \frac{1}{2} (m^2 - i\delta)^{\frac{D}{2}-1} \int_0^{\infty} dy y^{\frac{D}{2}-1} (y+1)^{-}1 \\
&= \frac{-i}{(2\pi)^D} \frac{2\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2})} \frac{1}{2} (m^2 - i\delta)^{\frac{D}{2}-1} B(\frac{D}{2}, 1 - \frac{D}{2}) \\
&= \frac{-i}{(2\pi)^D} \frac{2\pi^{\frac{D}{2}}}{\Gamma(\frac{D}{2})} \frac{1}{2} (m^2 - i\delta)^{\frac{D}{2}-1} \frac{\Gamma(\frac{D}{2})\Gamma(1 - \frac{D}{2})}{\Gamma(1)}
\end{aligned} \tag{35}$$

Using Euler Beta Function in Equation 35 that can prevent the divergent problem.[19]

A.2 Feynman Parametrisation

Feynman parametrisation makes integrating dominator to auxilary parameters for the simpler dominators. The basic concept of Feynman Parametrisation is this.

$$\frac{1}{AB} = \int_0^1 dx \frac{1}{(xA + (1-x)B)^2} = \int_0^1 dx \int_0^1 dy \frac{\delta(x + y - 1)}{(xA + yB)^2} \tag{36}$$

We can differentiate the B from Equation 36 several times:

$$\begin{aligned}
\frac{1}{AB^2} &= \int_0^1 dx \int_0^1 dy \frac{2y\delta(x + y - 1)}{(xA + yB)^3} \\
\frac{1}{AB^3} &= \int_0^1 dx \int_0^1 dy \frac{3y^2\delta(x + y - 1)}{(xA + yB)^4} \\
\frac{1}{AB^n} &= \int_0^1 dx \int_0^1 dy \frac{ny^{n-1}\delta(x + y - 1)}{(xA + yB)^{n+1}}
\end{aligned} \tag{37}$$

Equation 37 can prove below.

$$\frac{1}{A_1 A_2 \cdots A_n} = \int_0^1 dx_1 \cdots \int_0^1 dx_n \frac{(n-1)!\delta(x_1 + \cdots + x_n - 1)}{(x_1 A_1 + \cdots + x_n A_n)^n} \tag{38}$$

Equation 36 is $n = 2$ case and we multiply $\frac{1}{A_{n+1}}$ to Equation 38, we can get

$$\begin{aligned}
&\frac{1}{A_1 A_2 \cdots A_n} \frac{1}{A_{n+1}} \\
&= \int_0^1 dx_1 \cdots \int_0^1 dx_n (n-1)!\delta(x_1 + \cdots + x_n - 1) \frac{1}{(x_1 A_1 + \cdots + x_n A_n)^n A_{n+1}}
\end{aligned} \tag{39}$$

$$
\begin{aligned}
&= \int_0^1 dx_1 \cdots \int_0^1 dx_n (n-1)!\delta(x_1 + \cdots + x_n - 1) \\
&\int_0^1 dx \int_0^1 dy \frac{n y^{n-1} \delta(x+y-1)}{(x A_{n+1} + y(x_1 A_1 + \cdots + x_n A_n))^{n+1}} \\
&= \int_0^1 dx_1 \cdots \int_0^1 dx_n \int_0^1 dy \frac{n!\delta(x_1 + \cdots + x_n - 1) y^{n-1}}{((1-y)A_{n+1} + y\,(x_1 A_1 + \cdots + x_n A_n))^{n+1}} \\
&= \int_0^{\frac{1}{y}} dx_1 \cdots \int_0^{\frac{1}{y}} dx_n \int_0^1 dy \frac{n!\delta(y x_1 + \cdots + y x_n - y) y^n}{((1-y)A_{n+1} + y\,(x_1 A_1 + \cdots + x_n A_n))^{n+1}} \\
&= \int_0^{\frac{1}{y}} y dx_1 \cdots \int_0^{\frac{1}{y}} y dx_n \int_0^1 dy \frac{n!\delta(y x_1 + \cdots + y x_n - y)}{((1-y)A_{n+1} + (y x_1 A_1 + \cdots + y x_n A_n))^{n+1}} \\
&= \int_0^1 dz_1 \cdots \int_0^1 dz_n \int_0^1 dy \frac{n!\delta(z_1 + \cdots + z_n - y)}{((1-y)A_{n+1} + (z_1 A_1 + \cdots + z_n A_n))^{n+1}} \\
&= -\int_0^1 dz_1 \cdots \int_0^1 dz_n \int_1^0 dy' \frac{n!\delta(z_1 + \cdots + z_n + y' - 1)}{(y' A_{n+1} + (z_1 A_1 + \cdots + z_n A_n))^{n+1}} \\
&= \int_0^1 dx_1 \cdots \int_0^1 dx_{n+1} \frac{n!\delta(x_1 + \cdots + x_{n+1} - 1)}{(x_1 A_1 + \cdots + x_{n+1} A_{n+1})^{n+1}}
\end{aligned}
\tag{40, 41}
$$

If the parameters $x_1, x_2 \cdots$ are all positive, we can substitute the limits of integration from 1 to $\frac{1}{y}$ as we already use the property that in the Dirac Delta Function on the Equation 37.[20][21][9]

A.3 Bubble loop integral example

Example of Feynman integral to keep the understand easier, select the bubble case of Feynman Integral. This example and Primary Decomposition (Appendix A.4) has referenced from book of Weinzierl.[22]

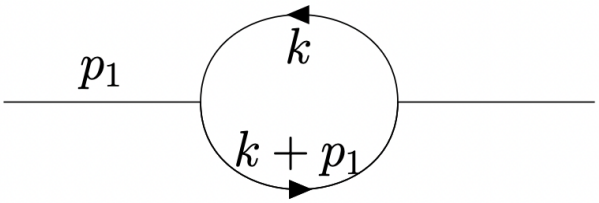


Figure 17: Bubble loop Feynman diagram

General form of integration of Bubble loop is under

$$
I_{bubble} = \eta^{2\epsilon} \int \frac{d^D k}{i\pi^{D/2}} \frac{1}{(k^2 - m^2 + i\xi)} \frac{1}{(k+p_1)^2 - m^2 + i\xi}
$$

In this example, $\nu = 2, \nu_1 = 1, \nu_2 = 1$
This bubble loop is 1-loop so $l = 1$ Use Feynman parameterize to I_{bubble}

$$
\begin{aligned}
I_{bubble} &= \eta^{2\epsilon} \frac{\Gamma(2)}{\Gamma(1)\Gamma(1)} \int_0^\infty dx_1 \int_0^\infty dx_2 \\
&\int \frac{d^D k}{i\pi^{D/2}} \frac{\delta(1 - x_1 - x_2)}{(x_1[k^2 - m^2 + i\xi] + x_2[(k+p_1)^2 - m^2 + i\xi])^2}
\end{aligned}
\tag{42}
$$

Expand denominator

$$\begin{aligned} &\eta^{2\epsilon}\frac{1}{1\cdot 1}\int_0^\infty dx_1\int_0^\infty dx_2 \\ &\cdot\int\frac{d^Dk}{i\pi^{D/2}}\frac{\delta(1-x_1-x_2)}{(x_1k^2-x_1m^2+x_1i\xi+x_2k^2+2x_2kp_1+x_2p_1^2-x_2m^2+x_2i\xi)^2} \\ =&...\int\frac{d^Dk}{i\pi^{D/2}}\frac{\delta(1-x_1-x_2)}{((x_1+x_2)k^2-(x_1+x_2)m^2+2x_2kp_1+x_2p_1^2+(x_1+x_2)i\xi)^2} \end{aligned} \tag{43}$$

These are the two graph polynomials. Substitution $k=\alpha k+\beta p_1$ and $k^2=\alpha^2k^2+2\alpha\beta kp_1+\beta^2p_1^2$ which can eliminate linear term in k which does not have spherical symmetry. And much easy to integrate.

$$\begin{aligned} &\frac{\delta(1-x_1-x_2)}{((x_1+x_2)\alpha^2k^2-2(x_1+x_2)\alpha\beta kp_1+(x_1+x_2)\beta^2p_1^2+2x_2\alpha kp_1+2x_2\beta p_1^2...} \\ &\overline{...+(x_1+x_2)m^2+(x_1+x_2)i\xi)^2} \end{aligned} \tag{44}$$

Compare to expression Equation 43 and Equation 44

$$\begin{aligned} (x_1+x2)\alpha^2(k^2)&=1 \\ ((x_1+x_2)2\alpha\beta+2x_2\alpha)(kp_1)&=0 \\ \\ \alpha^2&=\frac{1}{(x_1+x_2)} \\ \beta&=-\frac{x_2}{(x_1+x_2)} \end{aligned} \tag{45}$$

Therefore, insert the result α^2 and β in Equation 45 to Equation 44 and organize it

$$\begin{aligned} I_{bubble}=\eta^{2\epsilon}\int_0^\infty dx_1\int_0^\infty dx_2(x_1+x_2)^{D/2}\int\frac{d^Dk}{i\pi^{D/2}} \\ \frac{\delta(1-x_1-x_2)}{(k^2+[\frac{x_1x_2}{(x_1+x_2)}]p_1^2-(x_1+x_2)m^2+(x_1+x_2)i\xi)^2} \end{aligned} \tag{46}$$

Equation 43 used the property of dimensional regularisation called Scaling and Convert $[\frac{x_1x_2}{(x_1+x_2)}]p_1^2-(x_1+x_2)m^2=-R^2$ and put out delta function to outside integral form which is dependant on k.

$$\begin{aligned} I_{bubble}=\eta^{2\epsilon}\int_0^\infty dx_1\int_0^\infty dx_2(x_1+x_2)^{D/2}\delta(1-x_1-x_2)\int\frac{d^Dk}{i\pi^{D/2}} \\ \frac{1}{(k^2-R^2+(x_1+x_2)i\xi)^2} \end{aligned} \tag{47}$$

Use Equation 47 with one-loop master formula which is under

$$\int\frac{d^Dk}{i\pi^{D/2}}\frac{(-k^2)^a}{(-Uk^2+V)^\nu}=\frac{\Gamma(D/2+a)}{\Gamma(D/2)}\frac{\Gamma(\nu-D/2-a)}{\Gamma(\nu)}\frac{U^{-D/2-a}}{V^{\nu-D/2-a}} \tag{48}$$

Which is also express following set $a=0$, $U=1$, $\nu=N$ and $V=R-i\xi$

$$\int\frac{d^Dk}{i\pi^{D/2}}\frac{1}{(l^2-R^2+i\xi)^N}=(-1)^N\frac{\Gamma(N-D/2)}{\Gamma(N)}(R^2-i\xi)^{-N+D/2} \tag{49}$$

Use Equation 49 which is master formula to Equation 47[20][22]

$$\begin{aligned} I_{bubble} &= \eta^{2\epsilon}(-1)^2\Gamma(2-D/2)\int_0^\infty dx_1\int_0^\infty dx_2(x_1+x_2)^{D/2} \\ &\qquad \delta(1-x_1-x_2)(R^2-i\xi)^{-2+D/2} \\ &= \eta^{2\epsilon}\Gamma(2-D/2)\int_0^\infty dx_1\int_0^\infty dx_2(x_1+x_2)^{D/2}\delta(1-x_1-x_2) \\ &\qquad U^{D/2}(-p_1^2\frac{x_1x_2}{U}+\frac{m^2(x_1+x_2)^2}{U})^(-2+D/2) \\ &= \eta^{2\epsilon}\Gamma(2-D/2)\int_0^\infty dx_1\int_0^\infty dx_2\frac{U^{2-D}}{F^{2-D/2}}\delta(1-x_1-x_2) \end{aligned} \tag{50}$$

With $U = x_1 + x_2$, $F = -p_1^2 x_1 x_2 + m^2(x_1+x_2)^2$

Now we can expand the integral about $\epsilon = 0$(recall $D = 4-2\epsilon$) to $O(\epsilon^0)$and then integrate over x_1, x_2.
From Equation 50

$$I_{bubble} = \eta^{2\epsilon}\Gamma(2-D/2)\int_0^\infty dx_1\int_0^\infty dx_2\frac{(x_1+x_2)^{2-D}\delta(1-x_1-x_2)}{(-p_1^2x_1x_2+m^2(x_1+x_2)^2)^{2-D/2}} \tag{51}$$

insert $D = 4-2\epsilon$ and $x_2 = 1-x_1$ to Dirac delta function

$$I_{bubble} = \eta^{2\epsilon}\Gamma(\epsilon)(m)^{-2\epsilon}\int_0^1 dx_1(\frac{-p_1^2x_1(1-x_1)}{m^2}+1)^{-\epsilon} \tag{52}$$

Using
$\Gamma(\epsilon) = \frac{1}{\epsilon} - \gamma_E + O(\epsilon)$
$a^{-\epsilon} = exp(-\epsilon ln(a)) = 1 - \epsilon ln(a) + O(\epsilon^2)$
In here, $a = \frac{-p_1^2x_1(1-x_1)}{m^2} + 1$ and $(\frac{1}{\epsilon} - \gamma_E) = \frac{r_\gamma}{\epsilon} + O(\epsilon)$

$$\begin{aligned} I_{bubble} &= \eta^{2\epsilon}(\frac{1}{\epsilon}-\gamma_E)(m)^{-2\epsilon}\int_0^1 dx_1(1-\epsilon ln(\frac{-p_1^2x_1(1-x_1)}{m^2}+1))+O(\epsilon) \\ &= \eta^{2\epsilon}r_\Gamma(m)^{-2\epsilon}\int_0^1 dx_1(\frac{1}{\epsilon}-ln(\frac{-p_1^2x_1(1-x_1)}{m^2}+1))+O(\epsilon) \end{aligned} \tag{53}$$

Integral with Feynman Parameter x_1, it is quite messy in generally (Not in this example) and this is the main reason and result of this dissertation. For fast calculation it use NN.

$$I_{bubble} = \frac{\eta^2}{m^2}^{-\epsilon} r_\Gamma(\frac{1}{\epsilon}+2-\beta ln(\frac{-\lambda_+}{\lambda_-}))+O(\epsilon) \tag{54}$$

A.4 Primary Decomposition

From the bubble loop integral form which is tensor integral form also, it shows like under.

$$\begin{aligned} I_{bubble} &= \eta^{2\epsilon}\Gamma(2-D/2)\int_0^\infty dx_1\int_0^\infty dx_2\frac{U^{2-D}}{F^{2-D/2}}\delta(1-x_1-x_2) \\ &\qquad With U = x_1+x_2, F = -p_1^2x_1x_2+m^2(x_1+x_2)^2 \end{aligned} \tag{55}$$

After this result, it can be applied Primary Decomposition. Above equation can split 2 parts and eliminate the Dirac Delta Function which only remain the integrations are from 0 to 1.

$$\int_0^\infty dx_1 \int_0^\infty dx_2 = \sum_{l=1}^{2} \int_0^\infty d^2x \prod_{j=1}^{2} \theta(x_l \geq x_j) \tag{56}$$

In here θ-function defined as

$$\theta(x_l \geq x_j) = \begin{cases} 1 & \text{if x}\geq\text{y is true} \\ 0 & \text{otherwise} \end{cases}$$

In each domains it contains common factor and extract it

$$G = \eta^{2\epsilon}\Gamma(2 - D/2) \sum_{l=1}^{2} G_l$$

In this form integrals G_l we can substitute

$$x_j = \begin{cases} x_l t_j & \text{for } j < l \\ x_l & \text{for } j = l \\ x_l t_{j-1} & \text{for } j > l \end{cases} \tag{57}$$

then integrate out Feynman parameter x_l using Dirac Delta Function. All Feynman parameters are homogeneous (we do not define in this thesis). So U,F are homogeneous in L,$L+1$ respectively. It shows

$$\begin{cases} U(\overrightarrow{x}) \to U_l(\overrightarrow{t}) x_l^L \\ F(\overrightarrow{x}) \to F_l(\overrightarrow{t}) x_l^{L+1} 0 \text{ otherwise} \end{cases}$$

After all, using

$$\int \frac{dx_l}{x_l} \delta(1 - x_l(1 + \sum_{k=1}^{1} t_k)) = 1$$

derive

$$G_l = \int_0^1 \prod_{j=1}^{1} dt_j t_j^{2-1} \frac{U_l^{2-2D/2}(\overrightarrow{t})}{F_l^{2-D/2}(\overrightarrow{t})} \qquad l = 1, ..., N \tag{58}$$

This two (in this bubble loop) generated sectors called primary sectors. With primary sector decomposition we can derive different expression on $\int_0^\infty dx_1 \int_0^\infty dx_2$ to other.

$$\begin{aligned} \int_0^\infty dx_1 \int_0^\infty dx_2 = & \int_0^\infty dx_1 \int_0^\infty dx_2 \Theta(x_1 \geq x_2) \\ & + \int_0^\infty dx_1 \int_0^\infty dx_2 \Theta(x_2 \geq x_1) \end{aligned} \tag{59}$$

Each term is correspond to G_1, G_2 and $\Theta(x)$ is Heavyside function which could set the range. And insert it

$$G_1 = \int_0^\infty dx_1 \int_0^\infty dx_2 \frac{U^{2-D}(x_1, x_2)}{F^{2-D/2}(x_1, x_2)} \delta(1 - x_1 - x_2) \Theta(x_1 \geq x_2) \tag{60}$$

We can substitute $x_1 = x_1$ and $x_2 = x_1 t_1$ with $0 \leq t_1 \leq 1$ and it with Jacobian method derive $dx_2 = x_1 dt_1$

$$G_1 = \int_0^\infty dx_1 \int_0^1 dt_1 x_1 \frac{U^{2-D}(x_1, x_1 t_1)}{F^{2-D/2}(x_1, x_1 t_1)} \delta(1 - x_1 - x_1 t_1) \Theta(x_1 \geq x_1 t_1) \tag{61}$$

If the function is established like under, it means Homogeneous function

$$f(\lambda x_1, \lambda x_2, \lambda x_3) = \lambda^n f(x_1, x_2, x_3)$$

And $U = x_1 + x_2$, $F = -p_1^2 x_1 x_2 + m^2(x_1 + x_2)^2$ polynomials have homogeneity property so,

$$G_1 = \int_0^\infty dx_1 \int_0^1 dt_1 x_1 \frac{x_1^{2-D}}{x_1^{4-D}} \frac{U^{2-D}(1, t_1)}{F^{2-D/2}(1, t_1)} \delta(1 - x_1(1 + t_1)) \Theta(x_1 \geq x_1 t_1) \tag{62}$$

Using the Dirac Delta Function property which represent below,

$$\delta(g(x)) = \sum_i \frac{\delta(x - x_i)}{|g'(x_i)|}$$

where x_i are roots of $g(x)$.

So in the Equation 61, $\delta(1 - x_1(1 + t_1))$ and $1 - x_1(1 + t_1) = 0 \Rightarrow x_1 = \frac{1}{1+t_1}$ is a root. Therefore,

$$\delta(1 - x_1(1 + t_1)) = \frac{\delta(x_1 - \frac{1}{1+t_1})}{|-(1 + t_1)|} = \frac{\delta(x_1 - \frac{1}{1+t_1})}{1 + t_1}$$

it means that integrate 0 to ∞ in dx_1

$$\begin{aligned} G_1 = \int_0^\infty dx_1 \int_0^1 dt_1 \frac{1}{x_1} \frac{1}{1 + t_1} \frac{U^{2-D}(1, t_1)}{F^{2-D/2}(1, t_1)} \delta(x_1 - \frac{1}{1 + t_1}) \Theta(x_1 \geq x_1 t_1) \\ = \int_0^1 dt_1 \frac{U^{2-D}(1, t_1)}{F^{2-D/2}(1, t_1)} \end{aligned} \tag{63}$$

Note that δ sets $x_1 \to \frac{1}{1+t_1} \Rightarrow \frac{1}{x_1} \cdot \frac{1}{1+t_1} \to 1$
Same steps yield

$$G_2 = \int_0^1 dt_1 \frac{U^{2-D}(t_1, 1)}{F^{2-D/2}(t_1, 1)} \tag{64}$$

This process happens for all Feynman loop integrals because they are projective integrals (i.e the same property gives us the Cheng-Wu theorem[23])

$$I_{bubble} = \eta^{2\epsilon} \Gamma(2 - D/2)[G_1 + G_2] \tag{65}$$

Note that in G_1 we can set $U(t_1, 1) = 1 + t_1$ and $F(t_1, 1) = -p_1^2 t_1 + m^2(1 + t_1)^2$ so $x_1 \to 1$ and $x_2 \to t_1$
Also, in G_2 we can just set $x_2 \to 1$ and $x_1 \to t_1$

This bubble integral sector decomposition is not required since the integrand is already finite after primary decomposition. However, in general, we will encounter integrals with integrands similar example on chapter 1..

A.5 Sector Decomposition

ref.Hepp 66; Denner, Roth 96; Binoth, Heinrich 00 ref.Kaneko, Ueda 10; See also: Bogner, Weinzierl 08; Smirnov, Tentyukov 09

$$G_i = \int_0^1 dt_1 (\prod_{j=1}^{N-1} dx_j x_j^{\nu_j - 1}) \frac{U_i(\overrightarrow{x})^{expoU(\epsilon}}{F_i(\overrightarrow{x}, s_{ij})^{expoF(\epsilon)}} \tag{66}$$

In Equation 66 General form of Feynman integral have Polynomial terms like F and U which have overlapping singularities in the integral when the parameters x_j nears the 0.
For this technique iterated Sector Decomposition repeat[24]

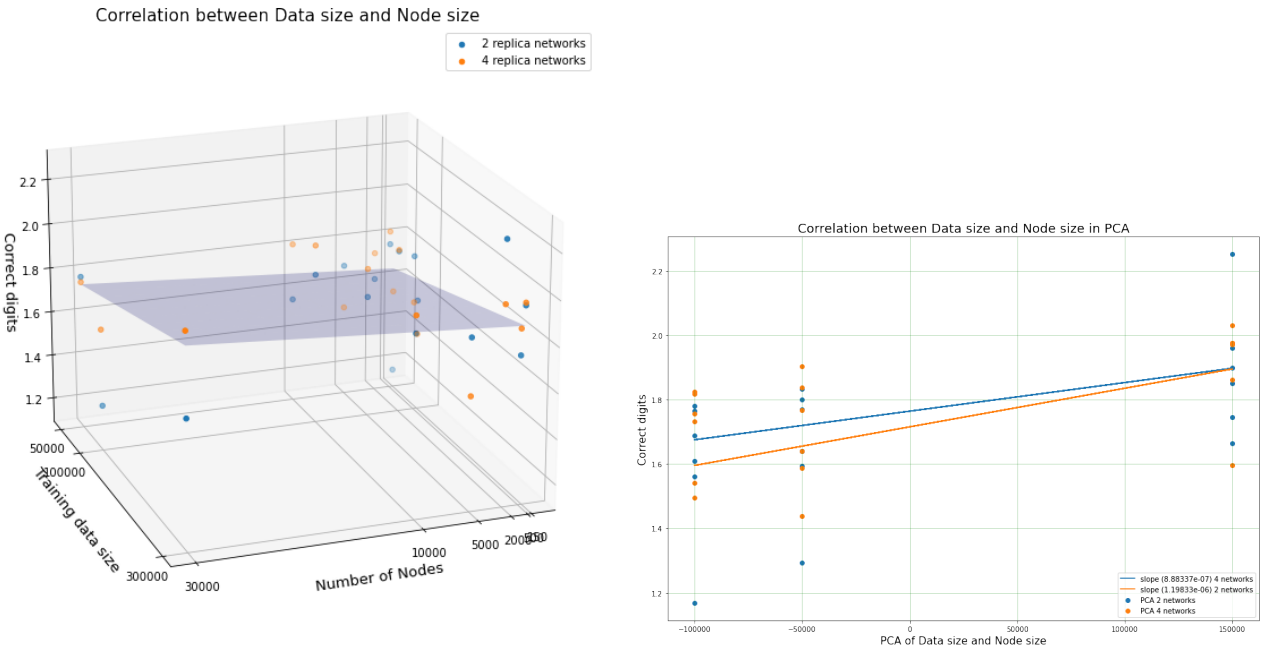
$$\begin{aligned} &\int_0^1 dx_1 \int_0^1 dx_2 \frac{1}{(x_1+x_2)^{2+\epsilon}} \\ &= \int_0^1 dx_1 \int_0^1 dx_2 \frac{1}{(x_1+x_2)^{2+\epsilon}} (\theta(x_1-x_2)+\theta(x_2-x_1)) \end{aligned} \tag{67}$$

In the Equation 67, singularities will overlap when the x_1 and x_2 goes to 0 it makes integral not completely. Therefore, we need to split the range which $x_1 > x_2$ and $x_1 < x_2$. And the function $\theta(x_i)$ must be positive.

$$\begin{aligned} &= \int_0^1 dx_1 \int_0^{x_1} dx_2 \frac{1}{(x_1+x_2)^{2+\epsilon}} + \int_0^1 dx_2 \int_0^{x_2} dx_1 \frac{1}{(x_1+x_2)^{2+\epsilon}} \\ &= \int_0^1 dx_1 \int_0^1 dt_2 \frac{x_1}{(x_1+x_1t_2)^{2+\epsilon}} + \int_0^1 dx_2 \int_0^1 dt_1 \frac{x_2}{(x_2t_1+x_2)^{2+\epsilon}} \\ &= \int_0^1 dx_1 \int_0^1 dt_2 \frac{x_1^{-1-\epsilon}}{(1+t_2)^{2+\epsilon}} + \int_0^1 dx_2 \int_0^1 dt_1 \frac{x_2^{-1-\epsilon}}{(t_1+1)^{2+\epsilon}} \end{aligned} \tag{68}$$

With Equation 68 singularities were factorised which means the integral value would be control by ϵ value. This strategy could be useful for general loops in Feynman diagrams.

A.6 Correlation of data size and number of nodes

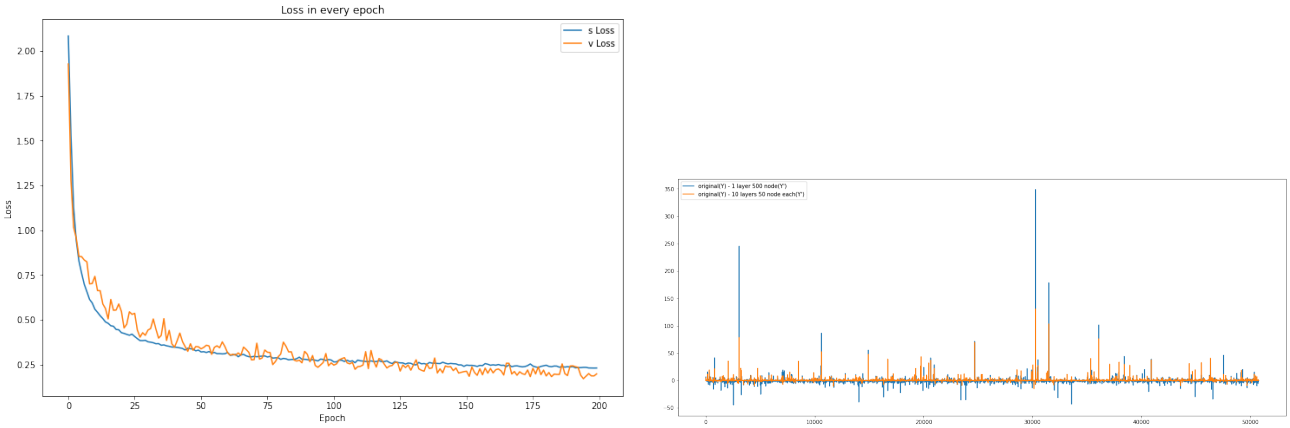


(a) (b)

Figure 18: (a) Relationship of data size and number of nodes in each network size. (b) Proceeding the PCA to clearly find out that data size and number of nodes have positive relation.

From the Figure 18, data size and number of nodes have positive relation. It means if the data size increase, number of nodes also increase. Therefore, both of elements in PNNI have correlation.

A.7 Brief reslut of layer-node trade off



(a) (b)

Figure 19: (a) Loss of every epoch, s loss has one layer v loss have 10 layers in network (b) Prediction of original dataset. Plot with closer to 0 have better result. Usually 10 layer network have better prediction.

Different functions may produce different results.

A.8 PNNI code

PNNI code git : `https://github.com/MinwooKim1990/disser/tree/main/Complete%20code`

References

[1] S. Lloyd R. A. Irani and M. Ahmadi. Using neural networks for fast numerical integration and optimization. *IEEE Access*, 8:84519–84531, May. 2020.

[2] P.van Dooren and L. de Ridder. An adaptive algorithm for numerical integration over an n-dimensional cube. *Journal of Computational and Applied Mathematics*, 3(2):207–217, 1976.

[3] S. Ferrari and R. F. Stengel. Smooth function approximation using neural networks. *IEEE Transactions on Neural Networks*, 16(1):24–38, Jan. 2005.

[4] S.S. Haykin. *Neural Networks: A Comprehensive Foundation 2Nd Ed.* Prentice-Hall Of India Pvt. Limited, 1999.

[5] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* Society for Industrial and Applied Mathematics, Jan. 1992.

[6] I.H. Sloan. Lattice methods for multiple integration. *J. Comput. Appl.Math*, 12-13, May. 1985.

[7] T. Gerstner and M. Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3):209–232, Mar. 1998.

[8] S.Nasuf. Exploring the prominent channel: Charged Higgs pair production in supersymmetric two-parameter non-universal Higgs model. *Nuclear Physics B*, 950:114847, 2019.

[9] J. M.Henn. Lectures on differential equations for Feynman integrals. *Journal of Physics A: Mathematical and Theoretical*, 48(15):153001, 2015.

[10] K. You M. Long, J. Wang and M. I.Jordan. How does learning rate decay help modern neural networks? [Online]. Available from: `http://arxiv.org/abs/1908.01878`, 2019. arXiv preprint arXiv:1908.01878.

[11] Quoc V.Le J. Ngiam A. Coates, A. Lahiri B. Prochnow and Andrew Y.Ng. On optimization methods for deep learning. *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML)*, pages 265–272, June. 2011.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[13] S. Borowka G. Heinrich M. Kerner S. Jahn, S. Jones and J. Schlenk. Numerical multiloop calculations: sector decomposition and QMC integration in pySecDec. *CERN Yellow Reports: Monographs*, 3:185–185, 2020.

[14] D. Nuyens and R. Cools. Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing Kernel Hilbert spaces. *Mathematics of Computation*, 75(254):903–920, 2006.

[15] J. Dick FY. Kuo and I.H. Sloan. High-dimensional integration: The quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 2013.

[16] H. Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of Number Theory*, 30(1):51–70, 1988.

[17] E. B. Baum and D. Haussler. What size net gives valid generalization? *Advances in neural information processing systems*, 1(1):151–160, 1988.

[18] J.F.Mas and J.J.Flores. The application of artificial neural networks to the analysis of remotely sensed data. *International Journal of Remote Sensing*, 29(3):617–663, 2008.

[19] J. C.Collins. *Renormalization: An introduction to renormalization, the renormalization group, and the operator-product expansion.* An Introduction To Renormalization, The Renormalization Group And The Operator-Product Expansion. Cambridge University Press, 1985.

[20] S. Weinzierl. Sector decomposition. *Geometric and topological methods for quantum field theory*, pages 144–187, 2010.

[21] K. Kannike. Notes on Feynman parametrisation and the Dirac delta function. [Online]. Available from: `https://kodu.ut.ee/~kkannike/english/science/physics/notes/feynman_param.pdf`, May. 2013.

[22] S. Weinzierl. Feynman integrals. [Online]. Available from: `http://arxiv.org/abs/2201.03593`, 2022. arXiv preprint arXiv:2201.03593.

[23] H. Cheng and T.T. Wu. Expanding protons: Scattering at high energies. *MIT Press, Cambridge, MA, USA*, pages 285–285, 1987.

[24] G. Heinrich. Numerical integration using sparse grids. *International Journal of Modern Physics A*, 23(10):1457–1486, 2008.