Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter now, including display on the World Wide Web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Yuwei Wu April 5, 2021

Physics-informed Deep Neural Networks for High Dimensional Integration

by

Yuwei Wu

Yuanzhe Xi
Advisor

Department of Mathematics

Yuanzhe Xi
Advisor

Manuela Manetta
Committee Member

Weihua An
Committee Member

2021

Physics-informed Deep Neural Networks for High Dimensional Integration

by

Yuwei Wu

Yuanzhe Xi
Advisor

An abstract of
a thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2021

Abstract
Physics-informed Deep Neural Networks for High Dimensional Integration
By Yuwei Wu

Monte Carlo methods are one of the most prevailing computational algorithms that use random sampling process to generate numerical estimations for problems in various fields, including mathematics, physical sciences, finance, etc. One of the applications of Monte-Carlo methods in mathematics is numerical integration, in which the speed of convergence is still slow. The Quasi-Monte Carlo method can improve the performance of the Monte Carlo method, especially in high dimensions, by generating low-discrepancy sequences. Existing Quasi-Monte Carlo methods use sequences such as Halton sequence, Sobol sequence, or Faure sequence that may fail to guarantee the low discrepancy due to improper predefined parameters. In this paper, we propose a different approach for generating low-discrepancy sequences through deep neural networks. By modeling the sequences as dynamic molecules and minimizing the total energy of the datasets in the deep neural networks, we are able to guarantee the low-discrepancy of the sequences that are independent of the initial distributions. We demonstrate the effectiveness of our methods using various numerical experiments for problems ranging from low to high dimensions.

Physics-informed Deep Neural Networks for High Dimensional Integration

By

Yuwei Wu

Yuanzhe Xi
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2021

Acknowledgements

I wholeheartedly thank Dr. Yuanzhe Xi for introducing me to the field of deep learning and providing me with invaluable feedback through this work. I would also like to thank Drs. Manuela Manetta, Weihua An for encouraging me and taking the time to serve on my thesis committee. I really appreciate Dr. Difeng Cai for his experience in Pytorch and neural networks. Finally, I would like to thank my family for supporting my studies and work at Emory.

Contents

Chapter 1 Introduction

Integration is the process of calculating the value of a specific integral by summing infinitesimal data over the domain. This process plays a very important role in the applications of mathematics and physics areas. A simple example of integration in physics can be given by integrating the acceleration function of an object with respect to its position, in which we get the function for the velocity of the object. Integration process like this serves as foundation for researchers to improve theories and applications in various fields.

1.1 Backgrounds

Consider a typical problem of evaluating the integral of a function f over a domain $D \subset \mathbb{R}^d$.

$$I = \int_D f(x)dx$$

When the integrand f admits some specific forms, one can find the analytical solution to the integration problem. However, in most cases, we will not be able to find the analytical solution for an integral but can only evaluate them at some points in the domain to obtain a numerical approximation for the integration result. This is because some of the functions f are too complicated to be evaluated analytically or because the dimension of the problem is too large. In some cases, quadrature rules such as trapezoid rule, Simpson's rule, or Gaussian quadrature are used to approximate the integration results for low dimensional problems. Results of these methods are largely based on the sample points we select over the domain. However, problems still arise when the dimension of the integration domain is large. This leads to an issue called "the curse of dimensionality," in which the number of estimations needed for evaluating the integral grows exponentially as the dimension

of problem increases [8]. Therefore, these quadrature methods are not suitable for high dimensional integration problems.

To approximate high dimensional integrals, Monte Carlo methods are one of the most prevailing classes of algorithms. In Monte Carlo integration, we produce a numerical result to approximate the integral value through generating a group of random samples over the domain of the integral. The convergence rate of Monte Carlo methods is independent of the dimensions involved in the problem, which makes it very efficient for solving high dimensional problems. The main motivation of this paper is to improve the efficiency of existing Monte Carlo integration methods for high dimensional problems via leveraging deep learning techniques.

1.2 Outline of the Thesis

In chapter 2, we review Monte Carlo methods that we plan to improve on. In chapter 3, we provide the background of energy potentials in physics and its connection to uniform distribution. In chapter 4, we review the basis of deep learning, and we propose a class of novel Quasi-Monte Carlo methods via deep neural networks in chapter 5. We demonstrate the viability and effectiveness of the proposed methods in chapter 6 through many numerical experiments and visualization graphs. Finally, we discuss the advantages of our methods in helping solve high dimensional problems in general and potential future work in chapter 7.

Chapter 2 Monte Carlo Methods

In this section, we briefly review the classical Monte Carlo methods and the Quasi-Monte Carlo methods. In particular, we focus on the advantages and disadvantages of these methods in evaluating high dimensional integration problems and the potential improvements on the existing methods. We also compare the convergence rates of these two classes of methods.

2.1 Monte Carlo Methods for Numerical Integration

Let's reconsider the previous stated problem of evaluating an integrand f over the domain $D \subset \mathbb{R}^d$.

$$I = \int_D f(x)dx. \tag{2.1}$$

If the dimension d of the domain is large, it is a common practice to use the Monte Carlo integration methods, which utilize a non-deterministic approach by averaging function values evaluated on randomly sampled points in the domain. Specifically, Monte Carlo integration methods generate a sample of size N in D, $x_1, \ldots, x_N$, and take the arithmetic mean of $\{f(x_k)\}_{k=1}^N$ as the final approximation for the integral result:

$$\int_D f(x)dx \approx \frac{1}{N}\sum_{k=1}^{N} f(x_k) = I_N[f]. \tag{2.2}$$

In this way, Monte Carlo integration methods can yield very precise estimates when the sample size of randomly generated samples is large. This is due to the law of the large numbers. We know that the expectation of Monte Carlo integration results is equal to the true value of the integral.

$$I = \lim_{N\to\infty} \frac{1}{N}\sum_{k=1}^{N} f(x_k). \tag{2.3}$$

In addition, the integration error of Monte Carlo methods can be defined as the following, which often serves as the measure in evaluating the accuracy level of different Monte Carlo integration results:

$$\epsilon_N[f] = I - I_N[f]. \tag{2.4}$$

Based on the Central Limit Theorem and the assumptions that the errors are uniformly distributed and have an expected value of zero, we can derive the variance and root mean square error (RMSE) which has the order of $O(\sigma N^{-1/2})$, where σ represents the constant variance of the integral f [2]. Therefore, we can see that the convergence rate of Monte Carlo methods based on uniform sampling is $O(N^{-1/2})$, meaning that if the sample size used in generating the integration results is four times the original size, the errors calculated based on Monte Carlo methods would be halved in general regardless of the dimension of the problem. This property reveals the most attractive feature of Monte Carlo integration methods: the convergence rate is independent of the dimension d [6].

One drawback, however, is that the speed of convergence of Monte Carlo method is still slow. A large number of samples is needed in order to achieve a relatively high accuracy level for the integration results. This slow convergence rate is mainly due to the use of random sampling. For a fixed number of samples, a better choice of samples may yield a much better accuracy in the final average estimation of the integration result. For example, in two dimensions with domain $D = [0, 1]^2$, a uniform tensor grid with N points yields a much higher accuracy than N random points sampled from the uniform distribution in D. Therefore, it is desirable for us to explore other methods to improve the convergence rate of Monte Carlo methods.

2.2 Quasi-Monte Carlo Methods

One of the variants of Monte Carlo methods is the Quasi-Monte Carlo methods, which improve the convergence rate of the classical Monte Carlo methods. Both methods evaluate an integral in a quite similar way, but Quasi-Monte Carlo methods generate N samples, $x_1, \ldots, x_N$, in D deterministically for integral calculations, which is often based on low-discrepancy theory.

Low-discrepancy sequence is different from the random sequence generated by Monte Carlo integration methods. For example, the discrepancy D_N associated with a sequence $x_1, \ldots, x_N$, in

D on the interval $[0, 1]$ is defined as follows [7]:

$$D_N = D_N(x_1, \ldots, x_N) = \sup_{0 \le \alpha \le \beta \le 1} |\frac{A([\alpha, \beta); N)}{N} - (\beta - \alpha)|,$$

where $A(E; N)$ be a counting function that is defined to be the number of terms in the finite sequence E and α and β are two arbitrary points in the interval $[0, 1]$.

Discrepancy for high dimensional sequences can be defined in similar ways. Based on this idea of discrepancy, we can say that for low discrepancy sequence, the proportion of points within a predefined set is proportional to the measure of this set over the entire domain. In this way, discrepancy can be used to measure the uniformity of a sequence distribution, and lower discrepancy means better uniformity. In Quasi-Monte Carlo methods, low-discrepancy or uniform sequences are often used in order to yield a faster speed of convergence to the integration results.

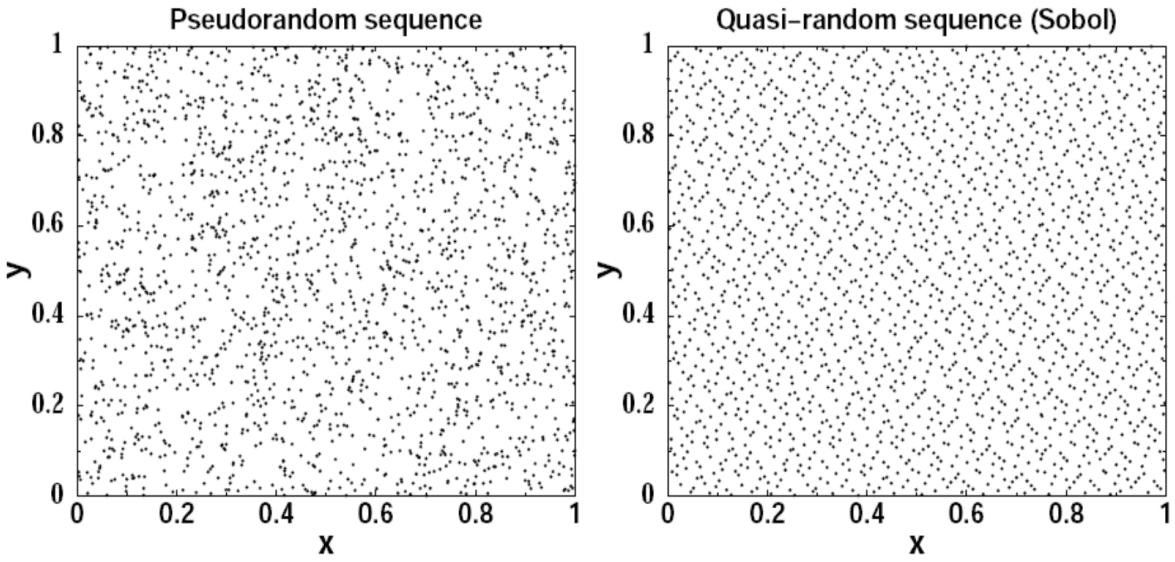


Figure 2.1: Pseudorandom Sequence and Quasi-random sequence [1].

Some common sequences used in Quasi-Monte Carlo methods include Halton sequence, Sobol sequence, and Faure sequence, all of which utilize one or two base numbers to form predefined sequences that are more uniformly distributed in different dimensions and help improve the integration accuracy [3]. As shown in the comparison in Figure 2.1, it is clear that randomly generated sequence tends to cluster together disorderly, whereas quasi-random sequence is distributed much more evenly and uniformly over the two dimensional domain, which leads to more efficient Monte

Carlo integration.

In Quasi-Monte Carlo methods, the convergence rate is roughly $O((logN)^d N^{-1})$ and is generally faster than the $O(N^{-1/2})$ convergence rate of regular Monte-Carlo methods [2]. Although Quasi-Monte Carlo methods enjoy a faster convergence rate in most cases, it is very difficult to evaluate the error bounds that depend on the sequence discrepancy in different approaches. Moreover, they may fail to guarantee the low discrepancy with poorly predefined parameters and when the dimension is extremely high.

After analyzing the regular Monte Carlo integration methods and the Quasi-Monte Carlo methods, we plan to overcome the drawbacks of Quasi-Monte Carlo methods and propose an efficient high dimensional integration scheme with strong theoretical guarantees. The idea of this new scheme comes from the energy potentials of physical particles which will be reviewed in the next chapter.

Chapter 3 Motivation from Physics

In this chapter, we try to explain our motivation in drawing the theoretical connection between energy potentials of physical particles in a system and the uniform distribution. Specifically, we analyze the notion of equilibrium state in physics from different perspectives and provide insights in utilizing the concept of equilibrium to model the sample points generated for the evaluation of Monte Carlo integration methods. We also introduce a specific distribution in explaining the intuition behind the proposed deep neural networks approach.

3.1 The Notion of Equilibrium

The science of statistical mechanics helps connect individual particles movement in a system to observable properties of the entire system [10]. One of the specific areas in statistical mechanics is the statistical thermodynamics, which uses overall physical properties (especially temperature or energy), rather than properties of individual particles, to describe the entire system. The concept of thermal equilibrium state in statistical thermodynamics refers to the situation when there is no exchange of energy or potentials among particles in an isolated system [18]. In our problem, we can treat the domain of our sampling space as an isolated system. Assuming that there is no energy exchange between the system and any external environment, we can reach the equilibrium state from the thermodynamics perspective when there is no energy transfer among all generated sample particles or no particle collision in our domain.

There is a similar concept of equilibrium in molecular dynamics, since different types of forces, either attractive or repulsive, exist among different molecules. When all forces are balanced out among all molecules in a system and the system would not tend to change over time, we know that the system reaches the equilibrium state from the perspective of molecular interactions. In

particular, we pay attention to the short-range repulsion forces between different molecules. Each molecule has its orbitals around the surface. When two molecules come too close to each other, there would be electrostatic repulsion forces between them, causing them to separate from each other [16]. This leads to a non-equilibrium change in state. Again if we treat our sampling space as a system consisting of molecules, to keep the entire system in a stationary state and achieve the equilibrium point, we need to make sure that all molecules are properly distanced from each other so that the forces can be minimized in the system.

Given these explanations of equilibrium, we can infer that little energy exchange and little interactions among microscopic particles in a system typically implies that the entire system is close to the equilibrium state. To prevent energy exchange, we can try to restrict particle collisions in the system or minimize the attractive and repulsive forces among different particle pairs by adjusting the locations and distance among all particles in the system. This inspires us to consider the uniform distribution as an equilibrium state, because in uniform distribution, distance among all particles is very well balanced so that there would be no collision or strong interactive forces among particles in the system. Therefore, we can model the randomly generated sample points in Monte Carlo methods as individual particles and the entire sampling domain as an isolated system. By moving the sample points to achieve better uniformity in the domain, we are able to minimize the energy exchange and particle interactions of the system and achieve a equilibrium state.

3.2 Boltzmann Distribution

There are other concepts that can well explain why lower energy implies more equilibrium and uniform distribution of particles. Boltzmann distribution is a famous probability distribution in statistical mechanics. This distribution draws the connection between the probability of a given state of the system and the total energy of the system. To be more specific, this probability distribution suggests that the probability of a system being in a certain state is inversely proportional to the energy of the system. This is the same as saying that it is more likely for us to find particles in the system being in lower energy states and gradually, the energy and momentum of different particles would converge to the uniform distribution in the given space [15].

The shape of Boltzmann distributions for molecules of ideal gas at different temperature may

be different, i.e. the probability distribution of the energy or speed of individual molecules in a given system may change given different temperature of the system. In general, we find that the number of particles with higher energy, which can be represented by the speed of particles, decreases exponentially in a system [14]. This corresponds to the notion of equilibrium state in which lower energy implies more stable and more stationary state of particles in the system. With minimum energy, the speed and momentum of individual particles in the system would all decrease to their minimum level and thus resemble a uniform distribution for energy of all particles.

Therefore, the Boltzmann distribution also suggests that trying to lower the energy of individual particles in a given system would help us effectively enter a more stable and stationary equilibrium state in the system with the distribution of the energy and position of particles resembling an uniform distribution. These ideas from thermodynamics and molecular interactions provide us a great way of modeling the sample points in the Monte Carlo methods and lead to our newly proposed neural networks approach in later chapters.

Chapter 4 Basics of Deep Learning

In this chapter, we introduce some basics of deep learning methods, including the multi-layer perceptron (MLP) and automatic differentiation (AD) technique. Both the MLP structure and the efficiency of AD technique stress the importance of utilizing deep learning approach to implementing our idea presented in chapter 3.

4.1 Multi-layer Perceptron

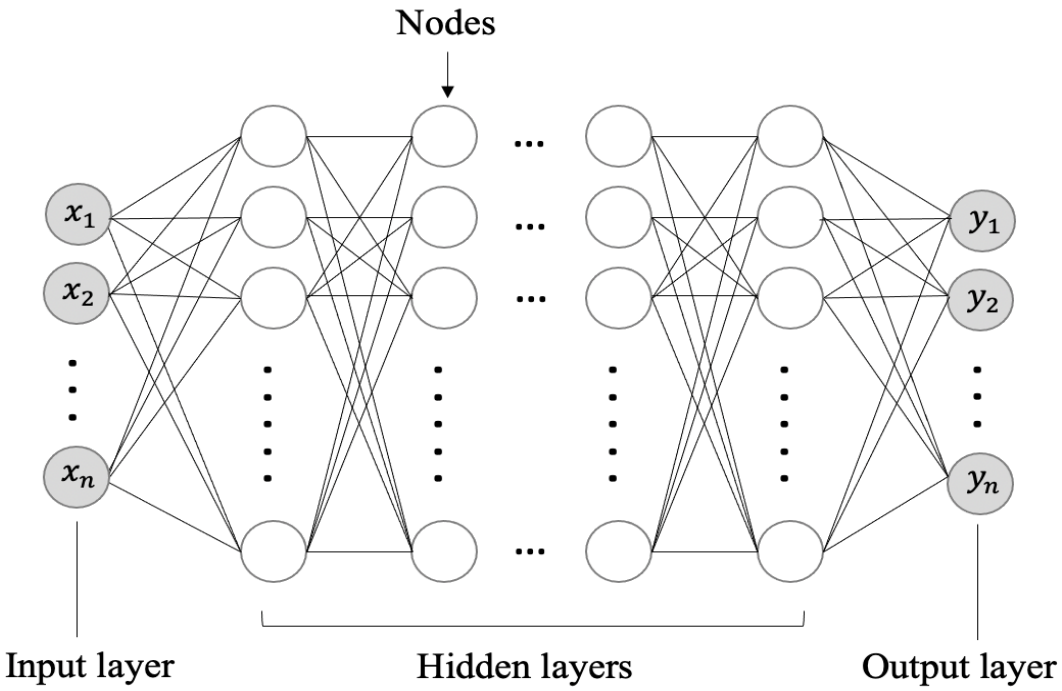


Figure 4.1: Deep Neural Network Structure.

A multi-layer perceptron (MLP) is a simple feedforward artificial neural network with the structure demonstrated in the figure 4.1. It is the most basic neural network structure and mainly

consists of three parts: the input layer, at least one hidden layer, and the output layer. Each node represents certain activation function that can perform calculations on the input data and pass the information to the following layers. Different layers are connected with different weights, which are trained and optimized during the model training process. The multiplayer perceptrons are very effective in detecting the features of input data and approximating any continuous function [11]. Due to the large sample sizes and high dimensions involved in in solving high dimensional integration problems, multi-layer perceptron becomes necessary for us in order to analyze each sequence distribution effectively and guarantee the low discrepancy in output sequence.

In our model, we mainly utilize the leaky ReLU and sigmoid activation functions in the nodes of our deep neural networks. First, we can add randomly generated samples $\{x_k\}$ into the the input layer of deep neural networks. These input data consist of arbitrarily artificial data points randomly generated in a chosen dimension and sample size. Multiple hidden layers in the neural networks would then help us detect the complex distributions of input sample points and minimize the total energy potentials and interactive forces of the input data, thus producing a low discrepancy sequence as the output of the neural networks. The neural network structure we used often needs to include more than 10 layers in order to process all data and minimize the total energy efficiently.

4.2 Automatic Differentiation

Another significance of using the deep learning approach to implement our idea presented in chapter 3 is that the automatic differentiation (AD) technique can be easily applied in the neural network algorithm.

Minimizing the total energy function and optimizing the locations of sample particles in the domain often requires huge computations of derivatives. Different methods that automatically calculate derivatives include finite differentiation, symbolic differentiation, and automatic differentiation. When the dimension of the problem is large, floating point errors exist in finite differentiation, and both the finite and symbolic differentiation algorithms become slow and memory-intensive. However, automatic differentiation helps differentiate mathematical functions numerically by a computer program consisting of elementary operations, which makes it highly efficient and applicable to even very complex functions [9].

Specifically, AD works in two modes: the forward mode and the reverse mode. For large sample size and high dimensional data, the reverse-mode AD is especially efficient. For each computation in the node, AD would calculate the corresponding gradient, evaluate it numerically, and combine the result with previously calculated results using chain rule. In this way, it can save much memory and computations when calculating derivatives and is especially easy to implement in the deep neural network algorithm with existing packages [9]. Details about our newly proposed deep learning approach are introduced in the next chapter.

Chapter 5 Dynamics, System Energy and Deep Neural Networks

To improve the convergence rate and robustness of Monte Carlo methods, we propose a new framework to generate sequences that can achieve better uniformity. Inspired by the equilibrium distribution in statistical mechanics and molecular interactions, we treat each sample as a particle of mass in the domain D and formulate the sampling problem as simulating the equilibrium state of a particle system. In this way, the sample sequences can be determined by searching for a distribution that minimizes certain energy potential functions of the particle system, where each candidate distribution can be represented by a deep neural network. This method seeks to maintain the convergence rate of Quasi-Monte Carlo methods while overcoming the drawbacks by guaranteeing the low-discrepancy of sequences generated.

We assume throughout the paper that the domain $D = [0, 1]^d$, where d is the dimension of the domain. As stated in chapter 3, to achieve the equilibrium state and better uniformity among all sample points, we need to make sure that there is no energy exchange and interactive forces among all samples, which means that all sample points should be well separated in the domain. Therefore, there should be no cluster among the sample points and no close pairs of sample points in the domain. In this case, the sample sequence would resemble the uniformly distributed sequence with low discrepancy and thus yield more accurate Monte Carlo integration estimations. Following this principle, we then propose two different approaches to achieve this equilibrium state in the system of sampling domain via deep neural networks.

5.1 Deep Learning Approach

Based on the concept of short-range repulsive forces in molecular interactions [16], closer pair of sample points implies larger energy. Therefore, the energy function that we aims to minimize in the deep neural networks should be a function of the distance between two different locations of sample points. Let's first define an energy function $U(x, y)$, which can measure the interaction forces between two particles located at x and y for $x \neq y \in \mathbb{R}^d$. In addition, let $V(x)$ be another potential function, such as a quadratic confining potential $V(x) = \frac{1}{2}|x - c|^2$ or a quartic confining potential $V(x) = \frac{1}{4}|x - c|^4$, etc., where c is the center position of the domain D. The potential function $V(x)$ can be viewed as a regularization term that prevents particles from going too close to the domain boundary. This type of regularization terms can serve different purposes. First, it can be used to restrict too much movement from the input sequence $\{x_k\}$ to the transformed sequence $\{y_k\}$ so that we may achieve a faster convergence to the integral. For example, this purpose can be achieved by including terms like $\sum_{k=1}^{n} ||y_k - x_k||_*$ for any specific norm or squared norm. Second, the regularization term can also help prevent the final Monte Carlo estimations using sequences generated from deep neural networks to get too far from the original Monte Carlo approximations. This can be achieved by using terms like $w_N|\sum f(y_k) - \sum f(x_k)|^2$ where w_N is a weight that depends on the sample size N. Since a larger N typically gives more accurate estimations in the classical Monte Carlo methods, it should be assigned a larger weight to prevent the final estimation from getting too different from the original estimation. In this way, $w_N \to \infty$ would yield the baseline Monte Carlo estimation results.

There are various forms for us to measure the energy potential $U(x, y)$ among a pair of sample points. Let $|x - y|$ denote the Euclidean distance between x, y and $||\cdot||$ denote a norm specified by the users. The energy function $U(x, y)$ can be a function such as a Coulomb interaction potential function $U(x, y) = \frac{1}{|x-y|}$, a Yukawa interaction $U(x, y) = \frac{e^{-\mu|x-y|}}{|x-y|}$ where μ is a damping factor, a negative distance function $U(x, y) = -||x-y||$, or a negative squared distance $U(x, y) = -||x-y||^2$, etc. Different forms of potential functions should be tested in order to decide which one is the most efficient and yield fast convergence to the integration results.

Given that $x_1, \ldots, x_N$ is a set of samples in domain $D = [0, 1]^d$, we aim to generate N new samples $y_1, \ldots, y_N$ in D, which achieves better uniformity than $\{x_k\}_{k=1}^N$. Instead of generating

uniform distributed sequence directly, we aim to improve the original poor distribution of the sample generated based on uniform sampling for regular Monte Carlo methods. This method can be summarized in the following way:

- Input: $x_1, \ldots, x_N \in D$
- Output: $y_1, \ldots, y_N \in D$
- Let $y = G(x; W)$ be a neural network with parameter W, where $G : D \to D$. We solve:

$$W^* = \arg\min_W \sum_{i \neq j} U(y_i, y_j) + \delta \sum_i V(y_i)$$

  where $\delta \geq 0$ is a regularization parameter;

- Define

$$y_k = G(x_k; W^*);$$

- Then the Monte Carlo approximation for integration estimation is given by

$$I_N[f] = \frac{1}{N} \sum_{k=1}^{N} f(y_k).$$

To further illustrate, this algorithm takes a set of sample sequence $\{x_k\}$ randomly generated in the domain D as input, and outputs a new set of points $\{y_k\}$ in the domain through running the deep neural networks and minimizing the total energy potentials of all sample pairs. Then we can evaluate the integral on the set of output sample points and take the average of all results as the final estimation using Monte Carlo integration methods. A simple example can be given in one dimensional case. If there is three sample points generated on the interval $[0, 1]$. An equidistributed sample set $\{0, 0.5, 1\}$ can minimize the short-range interaction among all sample pairs, because all points in this set are well-separated over the interval domain and it should also yield a very accurate result for the integral. This one dimensional example also coincides with some of the quadrature rules used in low dimensions.

Given this neural networks algorithm, we can then conduct numerical experiments in determining the optimal neural network structure and parameters, energy potential functions for U and V,

and the best accuracy results that our methods can achieve when producing numerical approximations of different integrals. In particular, we need to check whether the output set of sample points indeed follows a roughly uniform distribution and whether the final approximation for integrals using the output distribution is much more accurate than the original Monte Carlo methods.

Notice that in the stated algorithm, we consider mainly the short-range repulsion force between all the sample pairs, in which the closer the sample pair is, the larger the energy potentials between them. Theoretically, as long as no sample points cluster together, short-range repulsion potential is enough for us to generate a set of sample points that resembles the uniform distribution, so we can ignore the long-range attraction potentials between each sample pair here. Let us imagine that after we generate some random samples in the domain, there is still a large gap in the center of the domain, meaning that all sample points are somehow distributed evenly around the periphery. In this case, our deep neural networks would help us minimize the short-range repulsive energy by moving some sample points to the center of the domain so that energy near the periphery further decreases and so does the total energy of the sequence. Therefore, it is redundant for us to add long-range attraction potentials to attract points that are far away from each other to get closer. This may complicate the model too much and we also prove that this deep learning approach is efficient in improving results of existing Monte Carlo methods.

To recapitulate, our goal here is to minimize the function of total energy potentials among all sample value pairs generated in the domain via deep neural networks. Ideally, after training and running the deep neural networks repeatedly, we hope to reach the global minimum value of total energy potentials, which can help achieve the best uniformity among the sample points and yield the most accurate integration estimation.

5.2 Adaptive Training Approach

One of the potential drawbacks with the deep learning approach is that deep neural networks may incur huge computational costs, especially for high dimensional data or large sample size involved in the input data. Therefore, we try to utilize adaptive training to reduce time complexity and computational costs and improve the efficiency of the neural networks methods. The idea of using deep neural networks to minimize total energy of existing samples in the domain is retained.

The major difference between these two approaches is that we can add the randomly generated sample inputs step by step into the domain, instead of inputting all sample points at one time.

To be more specific, we can divide the total number of sample points into smaller groups with similar sample size at each group. First, given the predefined sample size, we generate the first sample group randomly over the domain. Second, we minimize the total energy of the first group of sample points so that all points are well separated and move to their optimal locations in this stage. The locations of the sample points in the first group are fixed afterwards. Repeatedly, we add the following groups with predefined sizes to the domain one by one and minimize the total energy at each group separately while considering all previous added sample points in the energy function as well. In this way, we again minimize the total energy at each step, which is dependent on all previous stages and has all previous added sample points held fixed in their optimal positions.

The goal of different approaches is the same. Given samples $x_1, \ldots, x_N$ in domain $D = [0,1]^d$, we aim to generate N new samples $y_1, \ldots, y_N$ in D, which achieves better uniformity than $\{x_k\}_{k=1}^N$. In the adaptive training, let's assume that we have initial m random samples, and at each stage we add random samples with a fixed size n to the domain D. After adding all the samples into the domain, we have N sample points in total. This method can be summarized in the following steps with similar deep neural networks structure as in section 5.1:

- Initial input samples: $x_1, \ldots, x_m \in D$ for $k = 1, \ldots, m$;
- Generate output $y_1, \ldots, y_m \in D$ such that

$$y_k = G(x_k; W^*),$$

  where

$$W^* = \arg\min_W \sum_{i \neq j} U(y_i, y_j) + \delta \sum_i V(y_i);$$

- Add another sample group with size n: $x_{m+1}, \ldots, x_{m+n} \in D$;
- Generate output $y_{m+1}, \ldots, y_m + n \in D$ such that W^* is minimized for $i, j = 1, 2, \ldots, m+n$ under the constraint that $y_1, \ldots, y_m$ are fixed;
- Repeat last two steps until the total number of samples reaches the desired sample size N.

In the adaptive training, deep neural network structure and parameters should be adjusted accordingly. For example, the number of layers in the neural networks and the number of epochs that neural networks need to run at each step can be reduced to proper sizes so that it is more suitable for smaller sample inputs. Because we minimize the total energy of samples with size m or n at each stage separately instead of minimizing total samples with size N, this adaptive approach can help us save much time and computational costs. However, although we may improve the efficiency of training neural networks, we may not be able to find the global minimum of total energy of all N samples points. Because we add randomly generated samples group by group, samples in some groups may cluster together in specific regions of the domain. With fixed locations of previously added sample points, movements of newly added points may be restricted around the local minimums of energy potentials. Therefore, although the adaptive approach may have faster convergence to the integration results, we give up finding the global minimum of total energy of all sample points in most cases. Without concerns in the computational costs, the standard deep learning approach is more desirable in order to achieve high accuracy levels in the integration approximation.

Chapter 6 Numerical Experiments

In this chapter, we conduct numerical experiments to test the validity and effectiveness of our methods. In particular, we test for different deep neural network architectures, energy functions U and V, dimensions d, integrands f, and input distributions x_k, etc. To test the accuracy of the output set of sample points in the Monte Carlo integration, we also generate integrals with varying parameters and their corresponding exact solutions. Then we compare the errors of the original Monte Carlo methods using the randomly generated sample points and errors of Monte Carlo methods using the output set of samples, i.e., errors calculated using $\{x_k\}$ and $\{y_k\}$ for $k = 1, 2, ..., N$, to test the accuracy levels of different approaches. More specific details of the numerical experiments are included in the following sections.

6.1 Energy Functions

Various potential functions that measure the interaction between two sample points can be considered in the experiments in order for us to figure out the most appropriate energy form for the loss function in the deep neural networks. Some of the common potential functions that we have considered in the experiments are listed here.

- $U(x, y) = \dfrac{e^{-\mu|x-y|}}{|x-y|^2}$ with parameter μ and

$$W^* = \arg\min_W \sum_{i \neq j} U(y_i, y_j)$$

- Coulomb potential[4] $U(x,y) = \frac{1}{|x-y|}$ plus quadratic potential $V(x) = \frac{1}{2}|x-c|^2$ with

$$W^* = \arg\min_W \sum_{i \neq j} U(y_i, y_j) + \delta \sum_{i=1}^{n} V(y_i)$$

  where δ is a regularization parameter.

- Yukawa potential [17] $U(x,y) = \dfrac{e^{-\mu|x-y|}}{|x-y|}$ with parameter μ and

$$W^* = \arg\min_W \sum_{i \neq j} U(y_i, y_j)$$

- Gaussian potential [12] $U(x,y) = e^{-\frac{|x-y|^2}{s^2}}$ with parameter s and

$$W^* = \arg\min_W \sum_{i \neq j} U(y_i, y_j)$$

In the experiments, we need to choose all parameters carefully so that the smaller the distance between each pair of sample points, the larger the energy potential between them. Different combinations of the potential U and V and potential functions with other forms can also be tested. Specifically, the Coulomb potential measures the electric energy between different electric charges, whereas the Yukawa potential measures the atomic interaction forces for particles within a short range. When the exponential term in the numerator of the Yukawa potential function goes to zero, the form of the potential function changes to the Coulomb potential. However, we find in the experiments that Coulomb potential is not very appropriate, because even when the sample points closely follow a uniform distribution, energy near the boundary of the domain is still very large, which is far from the minimum total energy that we try to find. As a conclusion, we find that when choosing parameters wisely, most potential functions can help reach our goal of achieving better uniformity in the output distribution. Among the functions we tested, Yukawa potential is the most effective one for our deep neural networks. Therefore, later examples all use the Yukawa potential function as the loss function.

6.2 Low Dimensional Examples

To illustrate the effectiveness of the deep neural networks approach, we generate some examples in lower dimensions, such as two dimensions and three dimensions for the visualization purpose. Since neural networks are related to dynamic systems, we can give a sequence of illustrations showing how $\{x_k\}$ changes into $\{y_k\}$ over time. In this way, we can also demonstrate how the total energy potentials of all sample points in the domain are minimized and how sample points gradually move into a much more uniform distribution.

The first example is the movement of 25 randomly generated sample points in the two dimensions. As previously mentioned, we choose the Yukawa potential form as the loss function, which is the most efficient one based on our experiments. Snapshots of the following scatterplots are taken at every 2000 epochs:

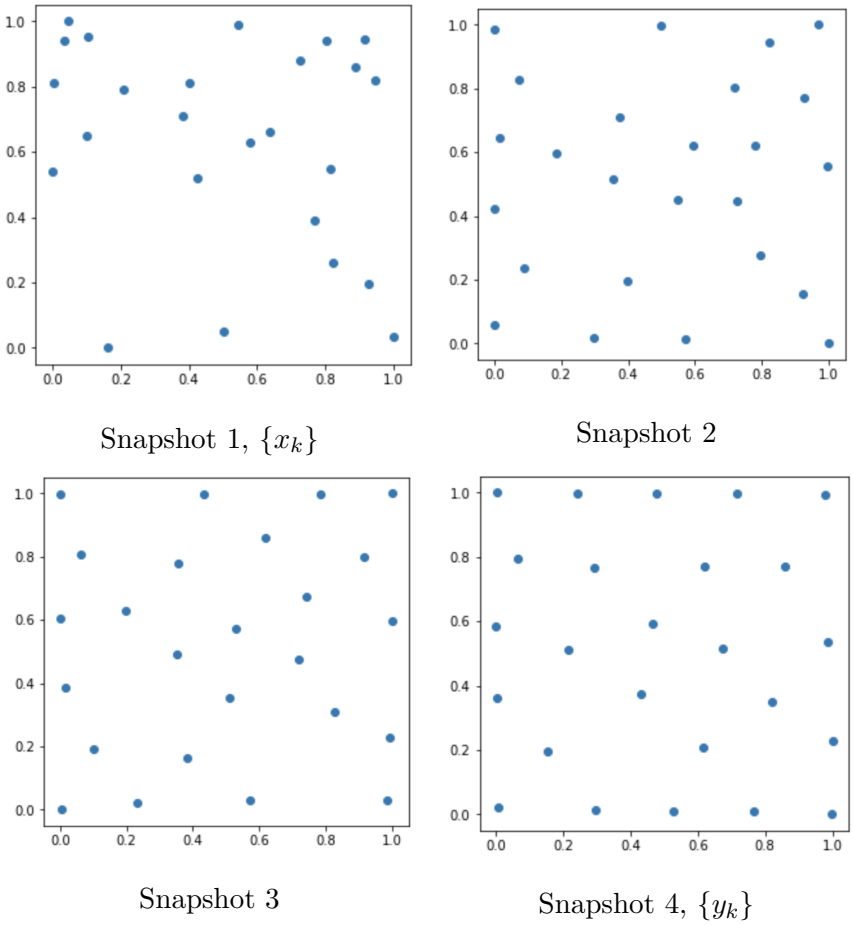
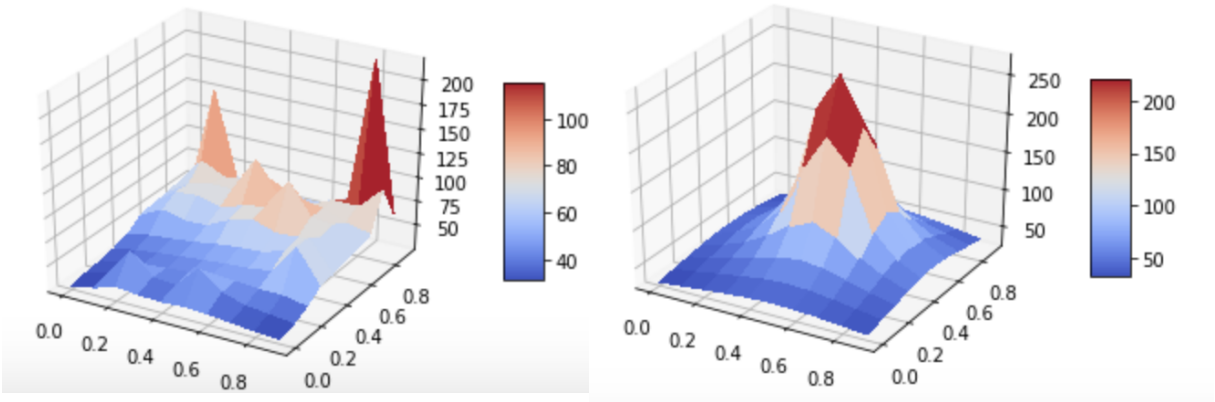


Figure 6.1: Snapshots of 25 Points in two dimensions.

As we can see, the initial distribution of the sample points $\{x_k\}$ for $k = 1, 2, \ldots, 25$ is highly unbalanced and asymmetric in the domain. Some points cluster together at the top right and top left corners. There are also several large gaps in the center and lower left corner of the two dimensional sampling space. After we finish tuning all parameters and running the deep neural networks repeatedly, these sample points start to move around their original places, and close sample pairs also spread out from each other gradually. By comparing the locations of sample points at each snapshot, we find that each point only takes a very small step around its original location towards the direction that minimizes the total potential energy of the set of samples. In the end, the output distribution of the set of 25 sample points $\{y_k\}$ greatly resembles the uniform distribution in the domain as shown in the snapshot 4, which can be used to effectively improve the accuracy of Monte Carlo results.



Heat Maps for Inputs

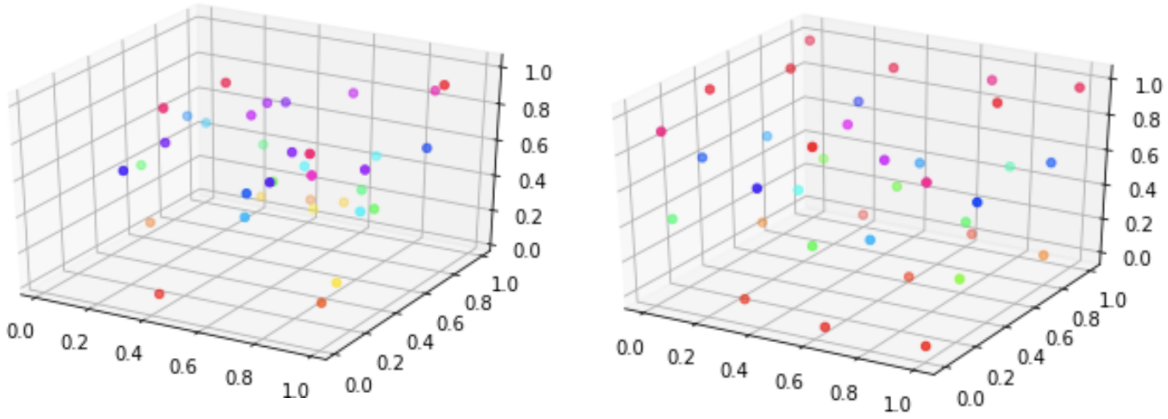
Heat Map for Outputs

Figure 6.2: Heat Map Comparisons.

A heat map can help us better illustrate the level of uniformity among the input and output sequences in the same two dimensional example. Specifically, let us assume the initial energy map is defined as $U_0(x) := \sum_{k=1}^{N} U(x, x_k)$ and the output energy map is given by $U_1(x) := \sum_{k=1}^{N} U(x, y_k)$, in which x represents every location in the two dimensional domain and the energy function U is represented again by the Yukawa potential function in this example. These energy maps can vividly exhibit the balance of energy potential at every location in the domain. To be more specific, the total energy concentrated at a specific location in the domain given its relationship to all sample data points is calculated. These energy maps help visualize the energy concentrated at specific

locations in the domain with different height and color, helping us discern the uniform distribution more efficiently.

As shown in the heap maps, we find that given the poor distribution of input data $\{x_k\}$, the energy map is highly unbalanced with high energy potential concentrating at the top right corner of the domain, which corresponds to the Snapshot 1 in the scatterplot illustrations. This type of heat maps implies a highly nonuniform distribution. On the other hand, in the output heat map, we find that it shows higher energy at the center of the domain and lower energy around the periphery. This is because center locations in the domain have closer distance to all sample points in sum and thus high energy to nearly all the generated sample points, whereas energy at locations near the periphery is only affected by several nearby sample points. Therefore, the second heat map gives us a good heat map representation of uniformly distribution sequences where the energy potential is balanced and almost symmetric with respect to all locations in the domain.



$\{x_k\}$ $\{y_k\}$

Figure 6.3: Inputs and Outputs of 36 points in three dimensions.

Similar illustrations can be recorded in the three-dimensional space. In this example, there are 36 sample points randomly generated in three dimensions, in which the color represents the height of the sample points. In the initial distribution, all randomly generated sample points seem to gather together near the center of the three dimensional domain and several pairs are very close to each other. After running our deep neural networks repeatedly, the set of output samples distribute much more evenly over the domain and are also well-separated from each other in any of the three dimensions. Utilizing the output samples $\{y_k\}$ of our neural networks instead of $\{x_k\}$

can significantly improve the original Monte Carlo results.

For the efficient adaptive training methods mentioned previously in Section 5.2 in which we minimize the total energy for all existing points in the domain step by step, we can also generate a sequence of output dynamics from the deep neural networks. For instance, randomly generated 56 sample points in the two dimensional domain may have a very poor distribution, as shown the scatterplots. Most of the randomly generated points cluster at the top of the domain, and several sample pairs are located too close to each other, leading to low accuracy of Monte Carlo results. To produce a low discrepancy sequence using the adaptive approach with our deep neural networks, we can generate an initial random sample group with size 16. After running 2000 epochs, the total energy of these initial 16 samples is minimized, so these initial sample points are distributed evenly in the two dimensional space, as demonstrated in the subfigure Stage 1 of Figure 6.5. Then we fix the locations of these initial 16 samples, add 10 more sample points at the next stage, and minimize the total energy of all existing sample points in the domain again. By adding 10 sample points at each stage and minimizing the total energy in the domain at each stage, we are able to move the added samples to their optimal positions at each stage. In the end, we are also able to generate a low-discrepancy sequence as the output sequences. As we can see in the last subfigure in Figure 8, although the output distribution $\{y_k\}$ still looks slightly unbalanced near the boundary of the domain, it has a much more uniform distribution than the input distribution indicated in Figure 7. In this adaptive approach, instead of minimizing the total energy of 56 sample points, we minimize 16 or 10 samples each time and optimize each group of samples based on previous conditions. Therefore, the computational costs of running our algorithm are also greatly reduced and the energy function would converge faster to the integration results.

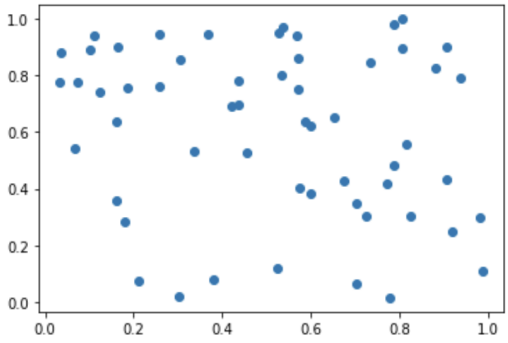


Figure 6.4: $\{x_k\}$

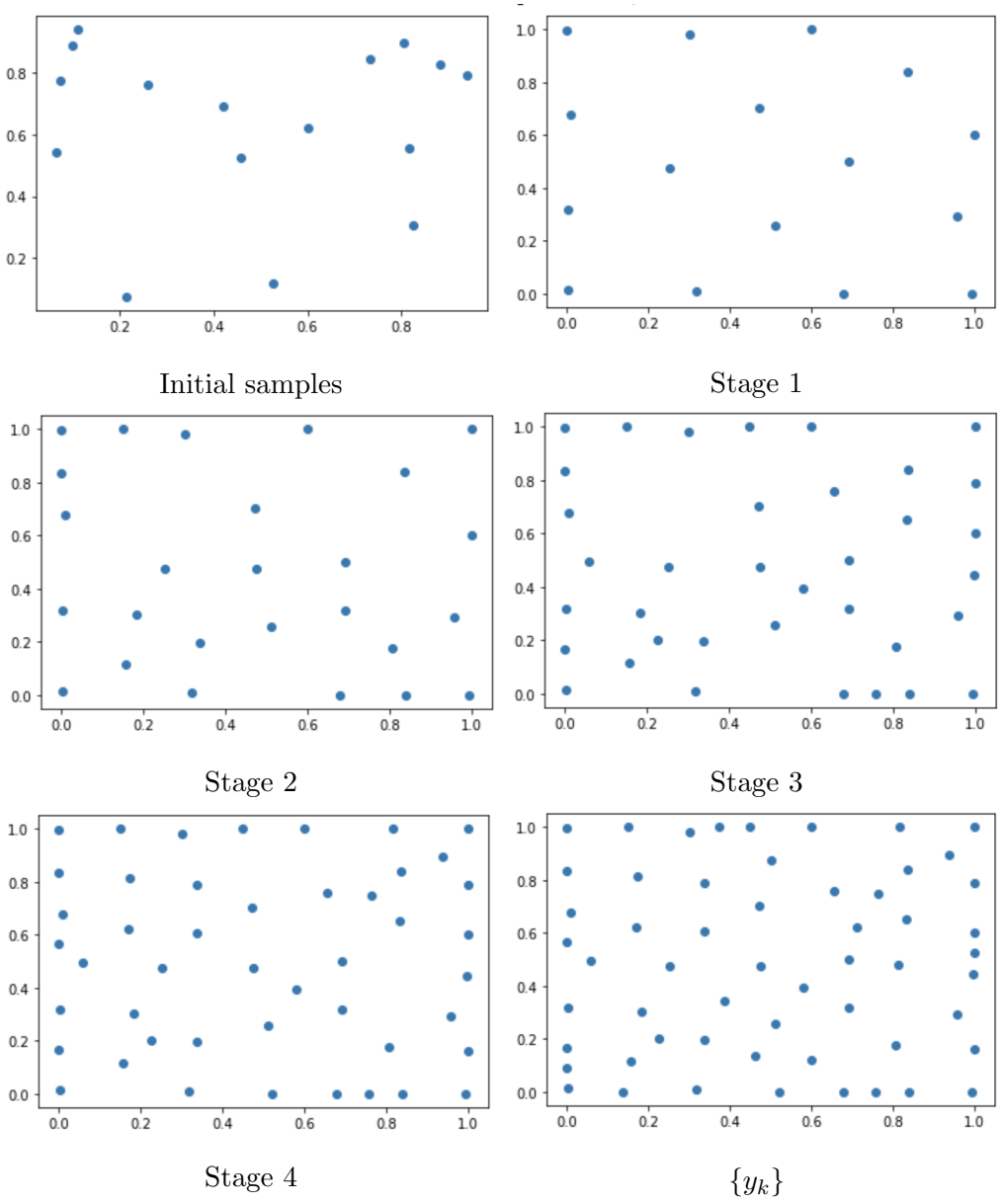


Figure 6.5: Adaptive Training in Generating Low-discrepancy Sequence

6.3 Results Evaluation

In order to clearly compare the accuracy levels of the original Monte Carlo integration methods and the deep neural networks based Monte Carlo methods, we choose integrands with known integration results and compare the errors in Monte Carlo results calculated using randomly generated samples $x_1, \ldots, x_N$ and deep neural networks outputs $y_1, \ldots, y_N$. We conduct numerical experiments and generate some sample data in the different dimensions.

Specifically, the test integrals we use in the experiments has the following form taken from Gantner & Schwab [5], in which d represents the dimension and c and b are parameters.

$$g(\mathbf{x}) = exp(c\sum_{j=1}^{d} x_j j^{-b}) = \prod_{j=1}^{d} exp(cx_j j^{-b}) \tag{6.1}$$

Since g is a product of one-dimensional functions, we can find out the exact solution of the one-dimensional integrals, which give us the exact value of the integral to compare with.

$$I(g) = \int_{[0,1]^d} g(\mathbf{x})d\mathbf{x} = \prod_{j=1}^{d} \frac{exp(cj^{-b}) - 1}{cj^{-b}}, c \neq 0 \tag{6.2}$$

Subscripts 1 and 2 in the tables represent errors calculated using two different test integrals, with parameters $c = 1, b = 2$ and $c = -1, b = 2$ respectively.

All sample results are generated using the Yukawa potential form as the energy function, and the parameter μ is chosen in a way that the larger the sample size or the dimension is, the larger the parameter μ is in the Yukawa form. In addition, average random errors are calculated by taking the average of errors calculated using random sample points $\{x_k\}$ generated with 10 different seeds and can be viewed as a good approximate for the general errors calculated using randomly generated samples.

Dimension d = 5				
Sample Size	Average Random Error_1	DNN Error_1	Average Random Error_2	DNN Error_2
200	0.04410483	0.008264	0.01055115	0.008505
500	0.03806009	0.006647	0.00804757	0.007664
1000	0.02332605	0.004515	0.00579373	0.003014
2000	0.01027604	0.002657	0.00271604	0.000461
5000	0.005875	0.000576	0.00144298	0.000295

Dimension d = 10				
Sample Size	Average Random Error_1	DNN Error_1	Average Random Error_2	DNN Error_2
500	0.03597211	0.005675	0.00817588	0.004594
1000	0.01888339	0.001806	0.0040605	0.004399
2000	0.01112589	0.001325	0.00289098	0.002894
3000	0.01121042	0.000918	0.00247913	0.00094
5000	0.00997146	0.000018	0.00182946	0.000426

Dimension d = 25				
Sample Size	Average Random Error_1	DNN Error_1	Average Random Error_2	DNN Error_2
800	0.01637029	0.00691	0.00342662	0.000623
1000	0.015864	0.003148	0.00360773	0.000403
2000	0.01696143	0.000811	0.00307998	0.000596
5000	0.00848933	0.0006	0.00165439	0.000136
7000	0.00762113	0.000341	0.00145294	0.00016

Dimension d = 50				
Sample Size	Average Random Error$_1$	DNN Error$_1$	Average Random Error$_2$	DNN Error$_2$
800	0.02360986	0.001241	0.00469252	0.001906
1000	0.02147316	0.000719	0.00395262	0.001701
2000	0.01291208	0.000507	0.00285905	0.000201
5000	0.01034117	0.000399	0.00218529	0.000385
8000	0.00742425	0.000135	0.00166327	0.000269

In these numerical experiments, our deep neural networks incorporate 10, 12, 13, and 15 different layers for dimensions of 5, 10, 25, and 50, respectively. First, we can easily find that both the average random errors and errors calculated based on deep neural networks decrease predominantly as the sample size increases. More sample data can surely yield preciser results based on the law of large number. Second, it is clear that in most cases, the deep neural network approach can achieve a three-digit or four-digit level of accuracy, which means that our methods can often yield one more digit of accuracy than original Monte Carlo integration methods.

In addition, although we test two different integrals in the experiments and errors calculated with randomly generated sample points are quite different (errors calculated for the second integral are much smaller than those calculated for the first integral in all presented dimensions), we find that errors calculated with the deep neural networks approach are similar. Take the fifth dimensional data as an example. The random errors generated with $\{x_k\}$ in the second integral are indeed much smaller than errors calculated for the first integral. However, in the deep neural network approach, errors calculated using two different integrals are quite close to each other. Although there are some numerical discrepancies, the fractional accuracy of these DNN errors are on the same level and this approach can always maintain the three-digit level of accuracy for both integrals. This phenomenon suggests that the global minimum energy may be independent of the initial distributions of randomly generated samples $\{x_k\}$. Regardless of the initial random distributions, 200 random samples in the fifth dimension can always find their optimal distribution in the domain, minimize the total energy function, and yield Monte Carlo results with errors of roughly 0.008. Undoubtedly, the process of minimizing the total energy can depend on the initial configuration,

since a better sampled sequence would yield a faster convergence and achieve the minimum energy with much fewer epochs. However, with properly tuned parameters and structure of deep neural networks, it is convincing that deep neural networks approach can guarantee a certain level of accuracy with a given sample size and dimension.

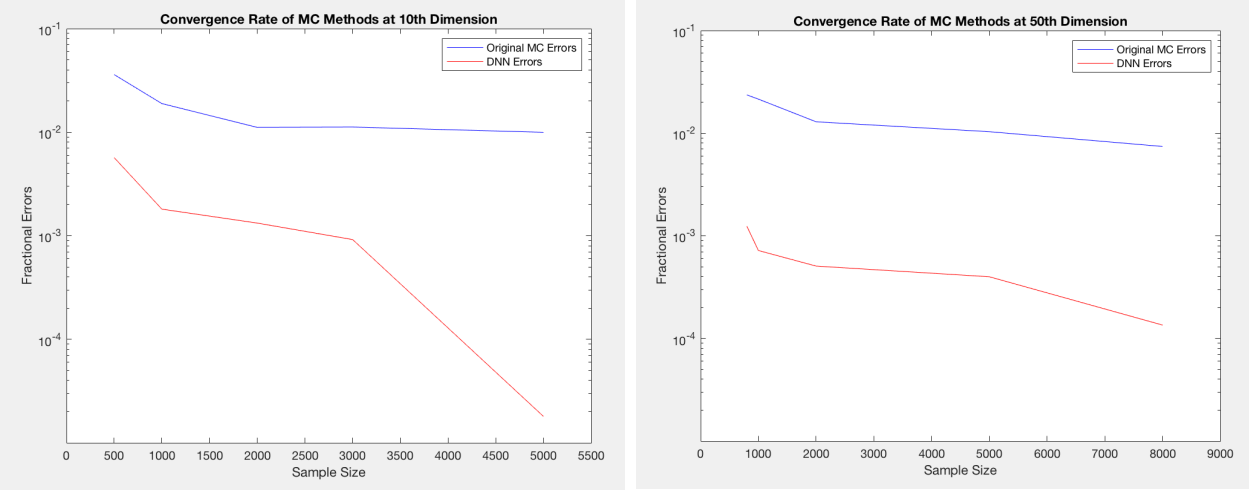


Figure 6.6: Convergence Rates at 10th and 50th Dimensions

We can compare the accuracy levels of both methods clearly using line graphs as illustrated. Here we compare two pairs of errors at the 10th and 50th dimensions as examples, and the y-axis is set at log scale. Both errors generated using original Monte Carlo methods and those produced using deep neural network approach decrease gradually as sample size increases. In particular, errors generated with the deep neural networks approach decrease at a much faster speed than errors of the original Monte Carlo methods at the 10th dimension and also slightly faster at 50th dimension. Although different parameters may produce different results and errors in each experiment, we can confidently conclude that the neural networks methods proposed in this paper can indeed help us improve the accuracy and efficiency of Monte Carlo methods for high dimensional data.

Together we can compare the errors we calculated from our deep learning approach in all chosen dimensions to find out the optimal level of accuracy that our methods can help achieve in Monte Carlo integration. As shown in the figure 6.7, errors generated with our deep neural networks seem to be independent of the dimension of the integration problem. We can always guarantee at least three-digit level of accuracy in the Monte Carlo integration results. As the sample size increases to roughly 2000, we can achieve a four-digit level of accuracy in most dimensions.

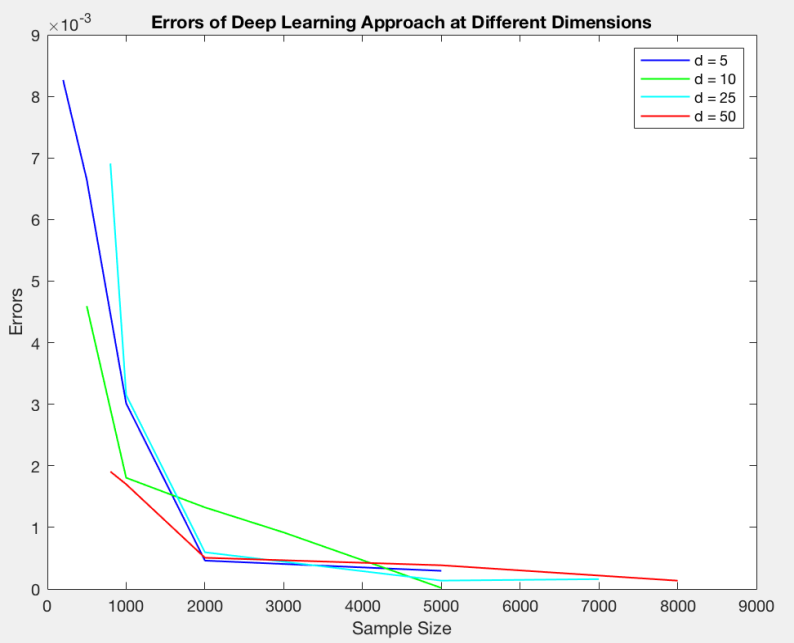


Figure 6.7: Lowest Errors of Deep Learning Approach at Different Dimensions

Chapter 7 Discussion and Future Work

Based on the neural networks methods proposed in this paper and numerical experiments of error comparisons for different approaches, we find that in many cases, our methods can achieve at least one more digit of accuracy than the regular Monte Carlo methods. In addition, errors calculated using the deep neural network approach also decrease at a much faster speed than the errors of the original Monte Carlo methods, which suggests that our approach can potentially improve the convergence rate of Monte Carlo methods from $O(\frac{1}{\sqrt{N}})$ to $O(\frac{1}{N})$ with properly chosen energy potentials functions and parameters. Although the initial distributions of randomly generated sample points seem to affect the speed of the decaying losses in our deep neural networks, the final results that minimize the total energy of all sample data are independent, meaning that the deep neural networks approach can guarantee a specific accuracy level for the Monte Carlo results regardless of the initial distributions of $\{x_k\}$. This also helps overcome the drawbacks of existing Quasi-Monte Carlo methods, which fail to guarantee the accuracy level in high dimensions. Our deep learning approaches have also contributed in the following ways:

General Tools for Improving Poor Distributions Because our deep neural networks can properly model the interactions among all the samples, these methods can be used as general tools to deal with various high dimensional problems, such as integration, sampling, inference, etc., by improving the poor distributions of input data. Ideally, after generating uniformly distributed sets of sample outputs by our neural networks, researchers can always transform the sequence and get the desired set of sample outputs or specific sample distribution with a given domain D, dimension d, and sample size N. Our neural network structure and energy potential functions provide a very convenient way of utilizing this type of tools.

New Definitions for Sequence Discrepancy Our methods belong to the broad class of Quasi-Monte Carlo methods which utilize the low-discrepancy of sequence, so we can come up with new definitions for sequence discrepancy based on the deep learning approach. For instance, because the energy potentials among the sample points are heavily affected by the distance among them, the distance between each pair of sample points in the domain plays a very important part in defining discrepancy. Let X be a set of sequence for $k = 1, \ldots, N$ in domain with dimension d. An example of the discrepancy of the sequence can be defined as:

$$\Lambda_N := \frac{\max\limits_{1\leq i\leq N} \delta_i}{\min\limits_{1\leq i\leq N} \delta_i}$$

where $\delta_i = \min\limits_{j\neq i} |x_i - x_j|$ denotes the Euclidean distance of x_i to the rest of the dataset X. This definition of discrepancy gives the ratio of the largest distance among sample data pairs to the smallest distance among sample pairs. Other similar definitions can also be derived from the neural networks perspective to better measure the discrepancy of a given sequence.

Admittedly, the deep learning implementation of the Quasi-Monte Carlo methods and the numerical experiments we have done on the deep learning approaches are highly limited. Future work should be conducted in the following directions to improve the proposed methods and their viability in real applications of Monte Carlo algorithms.

More Theoretical Justification In this paper, we have proven the viability and effectiveness of the deep learning approach and the adaptive training approach through numerical experiments and error comparisons with the standard Monte Carlo methods. More theoretical proofs are needed to verify that the minimum of total energy among all sample points always leads to low discrepancy in output sequence. Theories about different types of energy potential forms are also needed to prove the usefulness of each type in generating uniform distribution.

Function-dependent Sampling Given the effectiveness of our deep learning methods in improving poor distributions of sample data, future work can explore convenient ways of utilizing our neural network structure to implement function-dependent sampling. For example, researchers can

try to construct a library with proper parameters stored in it such that after users enter the domain D, dimension d, sample size N, and ideal function f, the deep neural networks can help generate a set of effective samples based on the underlying function.

Relation to Kinetic Monte Carlo Methods Kinetic Monte Carlo methods are stochastic simulation algorithms that simulate the evolution of some events occurring over time and have many applications in physics, including atom diffusion, atom disposition, or chemical reaction, etc. [13]. They utilize the transition rates of different states as inputs and try to model the dynamic behaviors of particles through individual steps. Given the similar dynamics of our approach inspired by molecule dynamics and the Kinetic Monte Carlo methods, future work can be done to explore how the idea of equilibrium state and our neural network structure can be exploited to improve the Kinetic Monte Carlo simulation.

Bibliography

[1] Juan A. Acebrón. Efficient methods for solving sdes. http://home.iscte-iul.pt/~jaats/myweb/Efficient%20methods%20for%20solving%20SDEs.html.

[2] Russel E Caflisch et al. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 1998:1–49, 1998.

[3] Kai-Tai Fang and Yuan Wang. *Number-theoretic methods in statistics*, volume 51. CRC Press, 1993.

[4] Richard Fitzpatrick. Coulomb's law. http://farside.ph.utexas.edu/teaching/em/lectures/node28.html.

[5] Robert N Gantner and Christoph Schwab. Computational higher order quasi-monte carlo integration. In *Monte Carlo and Quasi-Monte Carlo Methods*, pages 271–288. Springer, 2016.

[6] Malvin H Kalos and Paula A Whitlock. *Monte carlo methods*. John Wiley & Sons, 2009.

[7] Lauwerens Kuipers and Harald Niederreiter. *Uniform distribution of sequences*. Courier Corporation, 2012.

[8] Frances Y Kuo and Ian H Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52(11):1320–1328, 2005.

[9] Charles C Margossian. A review of automatic differentiation and its efficient implementation. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 9(4):e1305, 2019.

[10] Simon McQuarrie. Boltzmann factor and partition functions. https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/

Map%3A_Physical_Chemistry_(McQuarrie_and_Simon)/17%3A_Boltzmann_Factor_and_Partition_Functions.

[11] Tim Menzies, Ekrem Kocagüneli, Leandro Minku, Fayola Peters, and Burak Turhan. Using goals in model-based reasoning. *Sharing Data and Models in Software Engineering*, page 321–353, 2015.

[12] Halil Mutuk. Asymptotic iteration and variational methods for gaussian potential. *Pramana*, 92(4):1–5, 2019.

[13] Steve Plimpton. Kinetic monte carlo. https://cs.sandia.gov/~sjplimp/kmc.html.

[14] David Taylor. The maxwell-boltzmann distribution. https://faculty.wcas.northwestern.edu/~infocom/Ideas/mbdist.html.

[15] ZHENGQU WAN. The boltzmann distribution. 2017.

[16] Loren Dean Williams. Molecular interactions. https://ww2.chemistry.gatech.edu/~lw26/structure/molecular_interactions/mol_int.html#CVN.

[17] Hideki Yukawa. On the interaction of elementary particles. i. *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series*, 17:48–57, 1935.

[18] Bart Van Zeghbroeck. Statistical thermodynamics. https://ecee.colorado.edu/~bart/book/book/chapter1/ch1_4.htm#1_4_1.