NEURAL NETWORKS ARE INTEGRABLE

Yucong Liu
School of Mathematics
Georgia Institute of Technology
Atlanta, GA 30332
yucongliu@gatech.edu

ABSTRACT

In this study, we explore the integration of Neural Networks, a powerful class of functions known for their exceptional approximation capabilities. Our primary emphasis is on the integration of multi-layer Neural Networks, a challenging task within this domain. To tackle this challenge, we introduce a novel numerical method that consist of a forward algorithm and a corrective procedure. Our experimental results demonstrate the accuracy achieved through our integration approach.

Keywords Neural Network · Integration · Numerical Algorithm

1 Introduction

Deep learning models have demonstrated incredible power in various fields such as image and speech recognition, natural language processing, and autonomous driving in recent years. This work primarily centers on the Deep Neural Network, also known as multi-layer Perceptron or Feed-Forward Neural Network.

We begin by providing the definition of Neural Networks. A k-layer Neural Network ψ from $\mathbb{R}^{n_0}$ to $\mathbb{R}^{n_k}$ is a layer-wise structure. The i-the layer, for $i \in \{1, \cdots, k\}$, is defined as :

$$y^{(i)} = W^{(i)} x^{(i-1)} + b^{(i)}, \quad x^{(i)} = \sigma(y^{(i)}), \tag{1}$$

and $\psi(x) = W^{(k+1)} x^{(k)} + b^{(k+1)}$. In each layer, $W^{(i)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the weight matrix and $b^{(i)} \in \mathbb{R}^{n_i}$ is the bias vector. The input of $(i+1)$-th layer $x^{(i)} \in \mathbb{R}^{n_i}$ is the output of i-th layer, and $x^{(0)} = x$. Activation function σ is a nonlinear point-wise function, i.e $(\sigma(y^{(i)}))_j = \sigma(y_j^{(i)})$. One of the most well-known activation function is rectified linear unit (ReLU), which is defined as $\text{ReLU}(x) = \max(x, 0)$.

This work focuses on the topic of integration for both shallow and deep Neural Networks. Since gradient descent technique has been well used when optimizing a Neural Network, it's well studied in the literature. However the integrability of Neural Networks has garnered comparatively less attention. We aim to bridge this gap by providing explicit forms of integration for one-layer Neural Networks with any integrable activation function and deriving a piece-wise structure of the integration for multi-layer Neural Networks with ReLU activation function, along with a proposed algorithm with a corrector.

We will introduce our basic motivation in Section 2.1. For the integration of ReLU Neural Networks, we separate it into two cases, one-layer case in Section 2.2 and multi-layer case in Section 2.3. Our algorithm also works for Convolutional Neural Networks [Krizhevsky et al., 2017] and Residual Neural Networks [He et al., 2016], which will be introduced in Section 2.4. We will discuss about future work in Section 3 and present our experiments detail in Appendix C.

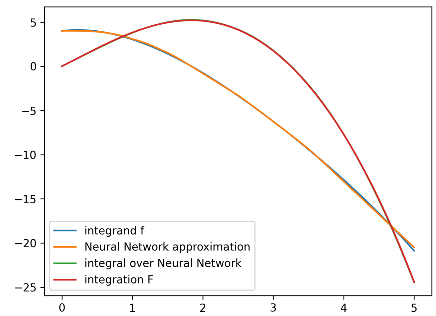
2 Integral and Algorithms

2.1 Basic Motivation

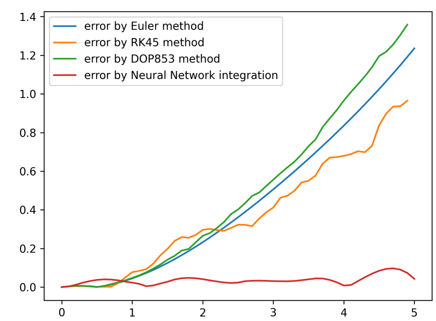
Let us begin by focusing on a classical numerical question: how can we approximate the integral of a function given only some samples? Assume $f : [a, b] \to \mathbb{R}$ is a integrable function, denote $F(x) = \int_a^x f(t)dt$ as the integral of function f. Suppose we have N samples of f, i.e., $\{(x_n, f(x_n))\}_{n=1}^N$. How can we recover F only with these samples without knowing the corresponding values $F(x_n)$ or the formula of f? Especially, can we get an accurate estimation for $F(b) = \int_a^b f(t)dt$?

Different from classic numerical integration algorithms, we here introduce one alternative way to solve this problem. With samples, we could approximate f using some integrable estimation $\hat{f}$, then approximate F by integral over $\hat{f}$. Universal Approximation Theorem [Cybenko, 1989, Hornik, 1991, Pinkus, 1999, Kidger and Lyons, 2020] guarantees that, for any $\varepsilon > 0$, there exists a Neural Network ψ, such that $|\psi - f| \leq \varepsilon/(b - a)$ on a compact set. Then the integral over ψ satisfies that $|\int_a^x \psi(t)dt - F(x)| \leq \varepsilon$, which gives a good estimation of function F. For high dimensional case, we consider the integral over a closed rectangle.

In the following part, our focus is on obtaining the closed-form solutions for integrating Neural Networks. We first provide explicit integration forms for one layer Neural Networks with any integrable activation function. Then, we derive a piece-wise structure for the integration of multi-layer Neural Networks that use ReLU activation function, and propose an forward integral algorithm with a corrector to enhance the accuracy of the integration.



(a) Neural Network approximation



(b) Absolute value of the approximation error given by different numerical integral algorithms.

Figure 1: Numerical Experiment

In Figure 1, we illustrate the approximation capabilities of a 2-layer Neural Network, showcasing that our numerical integral approximate perfectly for the integration F. Notably, our approach exhibits significantly smaller errors when compared to traditional numerical methods. For details about the experiments, please refer to Appendix C.

2.2 Integral over one-layer Neural Networks

We start with a trivial case: one layer Neural Networks $\psi : \mathbb{R} \to \mathbb{R}$

$$y = W^{(1)}x + b^{(1)}, \quad \psi(x) = W^{(2)}\sigma(y) + b^{(2)}. \tag{2}$$

Notice that in this one dimensional case, $W^{(2)}$ is a row vector with length n_1 and $W^{(1)} = (w_1, \ldots, w_{n_1})^\intercal$ is a column vector with length n_1. Then we state the following Lemma.

Lemma 2.1. *For a one-layer Neural Network ψ defined on a closed interval $[a, b]$, the integral of ψ can be expressed as:*

$$\int_a^x \psi(t)dt = W^{(2)}z + b^{(2)}(x - a),$$

where $z = (z_1, \ldots, z_{n_1})^\intercal \in \mathbb{R}^{n_1}$ and $z_i = \int_a^x \sigma(w_i t + b_i^{(1)})dt$.

Lloyd et al. [2020] focused on the integral of one layer Neural Network with logistic sigmoid activation function. We demonstrate that our analysis here works for general integrable activation function.

For n-dimensional case, the result of integral over rectangles is similar. Since one layer Neural Network is essentially a weighted sum of several integrable function, we only have to repeat Lemma 2.1 for n time. Let ψ be an one-layer Neural Network: $\mathbb{R}^n \to \mathbb{R}$. Then,

$$\int_{[a_1,b_1]\times\cdots\times[a_n,b_n]} \psi(x)dx_1 \ldots dx_n = \int_{[a_2,b_2]\times\cdots\times[a_n,b_n]} dx_2 \ldots dx_n \{W^{(2)}[\int_{a_1}^{b_1} \sigma(W^{(1)}x + b^{(1)})dx_1] + b^{(2)}(b_1 - a_1)\}. \tag{3}$$

As long as we have the explicit form of $\tilde{\sigma} = \int \sigma$, we can evaluate the integral directly. For instance, when ignoring the constant term,

$$\int \text{ReLU}(x) = \text{ReLU}^2(x)/2 \quad \text{and} \quad \int \tanh = \ln \cosh.$$

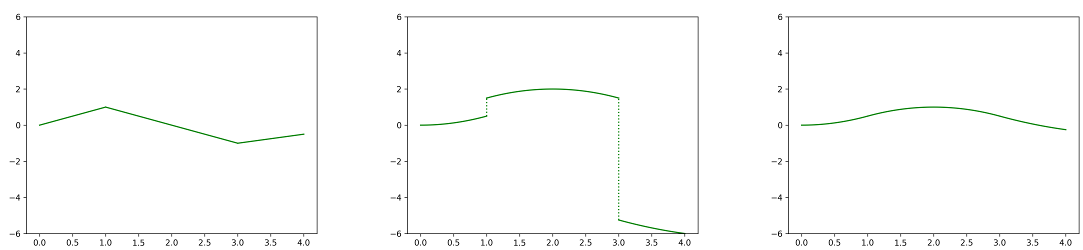
2.3 Numerical Integral over multi-layer Neural Networks

In this section, we will be delving into the integration of a multi-layer ReLU Neural Network, which is considerably more complex than the one-layer case. A single-layer Neural Network can be expressed as a simple weighted sum of several activation functions, making integration relatively straightforward. However, for a multi-layer Neural Network, the situation is quite different. Even with ReLU activation function, we do not possess explicit knowledge of the areas where the output of a neuron is zero or positive, particularly for neurons in higher layers. As a result, integration of multi-layer Neural Networks is a challenging task.

We start with an observation of ReLU Neural Networks. Arora et al. [2018] showed that every ReLU Neural Network is a piece-wise linear function, vice versa. We cite their Theorem here as reference.

Theorem 2.2 ([Arora et al., 2018])**.** *Every $\mathbb{R}^n \to \mathbb{R}$ ReLU Neural Network represents a piece-wise linear function, and every piece-wise linear function $\mathbb{R}^n \to \mathbb{R}$ can be represented by a ReLU Neural Network with at most $\lceil \log 2(n+1) \rceil + 1$ depth.*

Then to integral a ReLU Neural Network is equivalent with to integral a piece-wise linear function without explicitly knowing the break points. In each piece, the Neural Network can be represented in the form of $\alpha x + \beta$, where α and β are coefficients determined by the weights and biases in the network structure. Knowing these coefficients enables us to compute the integral of the Neural Network separately in each piece. Thus, we design a forward integral algorithm. We demonstrate that this algorithm works for batch of input and can be accelerated by GPUs.



(a) Piece-wise Linear Function (b) Integral in each piece (c) Correct the jump

Figure 2: Integration Process

However, by using a piece-wise integral, it is likely that we cannot obtain a continuous function since there are usually jumps at the breakpoints. Nonetheless, it is worth noting that these jumps are actually constant. To address this issue, we propose a numerical algorithm that corrects these jumps and ensures the continuity of the entire integral. Figure 2 illustrates the entire Neural Network integration process.

2.3.1 Forward Integral Algorithm

By Theorem 2.2, when we fix the input x, the output of j-th neuron in the i-th layer could be represented as a linear combination of input x plus a constant term, i.e $\alpha_{ij} x + \beta_{ij}$. Here α_{ij} is a vector and β_{ij} is a scalar. Then the output of $(i+1)$-th layer $\alpha_{i+1} x + \beta_{i+1} = \sigma(W^{(i+1)}(\alpha_i x + \beta_i) + b^{(i+1)})$. Here α_i is a matrix and β_i is a vector. Recall the definition of ReLU, it's equivalent with identity when the input is positive. Otherwise, it always outputs 0. As a result, given input x, the coefficient in the next layer depends on the coefficients in the current layer and whether each neuron passes through next layer, i.e

$$\alpha_{i+1} = \mathbb{1}_{\{x^{(i+1)}>0\}} \odot (W^{(i+1)} \cdot \alpha_i), \quad \beta_{i+1} = \mathbb{1}_{\{x^{(i+1)}>0\}} \odot (W^{(i+1)} \cdot \beta_i + b^{(i+1)}) \tag{4}$$

Here $\mathbb{1}$ denotes the indicator function and $\mathbb{1}_{\{x^{(i)}>0\}}$ represents which neuron passes to next layer, $\odot$ denotes Hadamard product, we use it to ignore neurons which stop. When computing, the Hadamard product between a vector and a matrix, it's implemented for each column of the matrix separately. Thus, following this procedure for each layer, we summarize our Forward Integral Algorithm 1 in Appendix A.

As long as we have α and β, we can directly get the integral at the piece where input x located in, i.e $\int_{a_i}^{x} \psi(x)dx_i$ can be represented a Polynomial term plus a constant term. Since the Polynomial term only depends on α and β, we denote it as Poly[α, β] for convenience. We demonstrate here, this algorithm is just a modified version of the forward algorithm to get output of a Neural Network by adding Hadamard product and indicator operators.

2.3.2 Numerical Corrector Algorithm

By Forward Integral Algorithm, we know the integral at the each piece while ignoring a constant term. Here, we introduce a numerical method to correct the jump at the break point between pieces.

Given an interval, we first select a partition $\{z_i\}_{i=1}^{N}$ of this interval. Denote the corresponding coefficient as $\alpha^{(i)}$ and $\beta^{(i)}$ for each z_i. Then we connect the integral at z_i by adding a constant term $\text{Poly}[\alpha^{(i-1)}, \beta^{(i-1)}](z_i) - \text{Poly}[\alpha^{(i)}, \beta^{(i)}](z_i)$ to make the integral continuous. The corrective steps are outlined in Algorithm 2 in Appendix B. It's worth noting that z_i can be conveniently chosen from samples x_i.

2.4 Extension to Convolutional Neural Networks and Residual Neural Networks

Convolutional Neural Networks [Krizhevsky et al., 2017] and Residual Neural Networks [He et al., 2016] are more powerful in practice. Our algorithms also work for them, when the activation function is ReLU.

The convolutional layer can be expressed as $\sigma(\kappa * x + b)$, where $*$ denotes the convolution operation, κ is the kernel and b is the bias vector. A convolutional Neural Network is one consist of convolutional layers and fully connected layers.

A Residual Block is usually consists of convolutional layers, fully-connected layers and a residual connection, i.e $\Psi(x) = x + \psi(x)$, where $\psi(x)$ can be expressed as a convolutional Neural Network. A Residual Neural Network is one taking structures of Residual Blocks, convolutional layers and fully-connected layers.

We first observe that the convolutional layer and the residual block would not affect the piece-wise structure for ReLU Neural Networks. As a result, our Forward Integral Algorithm works for both ReLU Convolutional Neural Network and ReLU Residual Neural Network. The convolutional layer works as same as the fully connected layer in our Forward Integral Algorithm because the convolution operator by kernel κ acts as a weighted linear combination of input. For Residual Neural Networks, we only have to add the coefficients at the beginning of Residual Block and the coefficients of ψ together due to residual connection. Details will be explained in Appendix A.

3 Discussion

We have developed novel algorithms to compute the integral of Neural Networks. In this section, we outline several promising directions for future research.

One immediate area of interest is the path integrals of Neural Networks, offering potential solutions to challenges in fields such as Molecular Dynamics [Marx and Parrinello, 1996, Li and Voth, 2022] and Quantum Mechanics [Feynman et al., 2010, HJ, 2012]. Our work could be beneficial for these scientific problems, as well as enhancing performance in vision tasks [Mildenhall et al., 2021].

Furthermore, Deep Ritz Method [E and Yu, 2018], Physics-Informed-Neural-Networks (PINNs) [Raissi et al., 2019, Pang et al., 2019, Mao et al., 2020, Karniadakis et al., 2021, Cai et al., 2021] and Neural Operators Li et al. [2021], Lu et al. [2021], Kovachki et al. [2023] have garnered significant attention. While current approaches focus on using derivatives of Neural Networks to satisfy partial differential equations (PDEs), exploring the integration of Neural Networks may open up new avenues for addressing these problems.

Neural Networks have proven effective in addressing challenges associated with density estimation and Bayesian inference, as evidenced by prior research [Magdon-Ismail and Atiya, 1998, Alsing et al., 2019, Lueckmann et al., 2019]. The integrability of Neural Networks holds the promise of substantial improvement in these applications, such as expectation and variance estimation, and entropy estimation, thereby enhancing their statistical perspectives [White, 1989, Cheng and Titterington, 1994].

Our algorithm depends on how well a Neural Network is trained. So, investigating the error bounds for these Neural Network based numerical algorithm adds an intriguing dimension to our exploration. Understanding the theoretical limits of approximation accuracy [Barron, 1994, Ronen et al., 2019, DeVore et al., 2021] and the factors influencing error is pivotal for assessing the reliability and robustness of such algorithms.

Acknowledgments

There is no funding to be disclosed.

References

Justin Alsing, Tom Charnock, Stephen Feeney, and Benjamin Wandelt. Fast likelihood-free cosmology with neural density estimators and active learning. *Monthly Notices of the Royal Astronomical Society*, 488(3):4440–4458, 2019.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding Deep Neural Networks with Rectified Linear Units. In *International Conference on Learning Representations*, 2018.

Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14:115–133, 1994.

Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021.

Bing Cheng and D Michael Titterington. Neural networks: A review from a statistical perspective. *Statistical science*, pages 2–30, 1994.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021.

John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Weinan E and Bing Yu. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications in Mathematics and Statistics*, 6(1):1–12, Mar 2018.

Leonhard Euler. *Institutionum calculi integralis*, volume 4. impensis Academiae imperialis scientiarum, 1845.

Richard P Feynman, Albert R Hibbs, and Daniel F Styer. *Quantum mechanics and path integrals*. Courier Corporation, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

W HJ. *Introduction to Quantum Mechanics: Schrodinger Equation and Path Integral*. World Scientific, 2012.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, Jun 2021.

Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Chenghan Li and Gregory A Voth. Using machine learning to greatly accelerate path integral ab initio molecular dynamics. *Journal of Chemical Theory and Computation*, 18(2):599–604, 2022.

Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021.

Steffan Lloyd, Rishad A. Irani, and Mojtaba Ahmadi. Using neural networks for fast numerical integration and optimization. *IEEE Access*, 8:84519–84531, 2020.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, Mar 2021.

Algorithm 1 Forward Integral Algorithm

Input: $x \in \mathbb{R}^n$, k-layer Neural Network with weights $W^{(i)}$, bias $b^{(i)}$
Output: First order coefficient α and constant term β
$\alpha \leftarrow I$
$\beta \leftarrow \mathbf{0}$
$y \leftarrow x$
for $i = 1$ **to** $k+1$ **do**
  if $i == k+1$ **then**
    $\alpha \leftarrow W^{(k+1)} \times \alpha$
    $\beta \leftarrow W^{(k+1)} \times \beta + b^{(k+1)}$
  else
    $y \leftarrow \sigma(W^{(i)} y + b^{(i)})$
    $z \leftarrow \mathbb{1}_{\{y>0\}}$
    $\alpha \leftarrow z \odot (W^{(i)} \times \alpha)$
    $\beta \leftarrow z \odot (W^{(i)} \times \beta + b^{(i)})$
  end if
end for
Return: α and β

Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H Macke. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pages 32–53. PMLR, 2019.

Malik Magdon-Ismail and Amir Atiya. Neural networks for density estimation. *Advances in Neural Information Processing Systems*, 11, 1998.

Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

Dominik Marx and Michele Parrinello. Ab initio path integral molecular dynamics: Basic ideas. *The Journal of chemical physics*, 104(11):4077–4082, 1996.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: representing scenes as neural radiance fields for view synthesis. 65(1):99–106, dec 2021.

Guofei Pang, Lu Lu, and George Em Karniadakis. fPInns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica*, 8:143–195, 1999.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Basri Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *Advances in Neural Information Processing Systems*, 32, 2019.

Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*, volume 375. Springer Berlin Heidelberg New York, 1996.

Halbert White. Learning in artificial neural networks: A statistical perspective. *Neural computation*, 1(4):425–464, 1989.

A Appendix: Forward Integral Algorithm

We present our Pseudo code for Forward Integral Algorithm 1 here. Notice that, the last layer is a little different since there is no activation function. Since $x = I \cdot x + \mathbf{0}$, we initialize α to be identity matrix and β to be zero vector $\mathbf{0}$.

For Residual Neural Network, the algorithm works with little difference. Suppose there is a residual connection after several layers, we can formulate is as $\Psi(x) = \phi(x) + \psi(\phi(x))$, where ϕ represents the previous layers, and ψ is a Convolutional Neural Network. Fix a input x, denote the coefficients for ϕ as α_ϕ, β_ϕ and the coefficients for $\psi(\phi(x))$ as α_ψ and β_ψ. Notice that these coefficients can be simply by our Forward Integral Algorithm 1. We only have to sum them up. Then the coefficients for Ψ is just $\alpha_\phi + \alpha_\psi$ and $\beta_\phi + \beta_\psi$.

Algorithm 2 Numerical Corrector

Input: Partition $\{z_i\}_{i=1}^N$ **on interval** $[a, b]$**, k-layer Neural Network** ψ
Output: $\int_a^b \psi dx_j$
$\alpha_0 \leftarrow 0$
$\beta_0 \leftarrow 0$
$C_0 \leftarrow 0$
for $i = 1$ **to** N **do**
    $\alpha_i, \beta_i \leftarrow$ Forward Integral Algorithm(z_i)
    $C_i \leftarrow$ Poly$[\alpha_{i-1}, \beta_{i-1}](z_i) -$ Poly$[\alpha_i, \beta_i](z_i) + C_{i-1}$
end for
Return: Poly$[\alpha_N, \beta_N](z_N) -$ Poly$[\alpha_0, \beta_0](z_0) + C_N$

B Appendix: Corrector Algorithm

Our Corrector Algorithm 2 lays a crucial role in refining the integral approximation, ensuring continuity at breakpoints within the piece-wise structure. The whole process works as accumulate the constant term through each partition point.

C Appendix: Experiment

We constructed extensive experiments to demonstrate that our algorithms work well on numerical integral, compared with traditional numerical algorithms.

We set the domain to be $[0, 5]$, $f(x) = \cos(x) - x^2 + 4 - 1/(x+1)$. Then the integration $F(x) = \int_0^x f(t)dt = \sin(x) - 1/3 * x^3 + 4x - \log(x+1)$. We trained a 2-layer Neural Network with width 100 to approximate f with 51 data points $\{x_i, f(x_i)\}_{i=0}^{50}$, where x_i evenly spaced over $[0, 5]$. We set epochs to be 200, learning rate 0.001 with batch size 20, using Stochastic Gradient Descent and mean squared error loss. When implement our Corrector algorithm, we set partition points z_i to be x_i.

For comparisons, we also implemented Euler-Forward method [Euler, 1845], Explicit Runge-Kutta method of order 5(4) (RK45) [Dormand and Prince, 1980] and Explicit Runge-Kutta method of order 8 (DOP853) [Wanner and Hairer, 1996].

We present our experiment results in Figure 1. In the top figure, we display our Neural Network's approximation capabilities. The 2-layer Neural Network exhibits outstanding approximations of the integrand f, and its integral simultaneously provides an excellent approximation of F. In the bottom figure, we track the approximation error $\left|F(x) - \hat{F}(x)\right|$, the absolute value of difference between true value and estimation. Here $\hat{F}(x)$ represents the estimated integral obtained through numerical algorithms. Notably, the integral computed by the Neural Network displays the smallest approximation error, highlighting its superior performance in our experiment.