Using Neural Networks for Fast Numerical Integration and Optimization

STEFFAN LLOYD\textsuperscript{\textcopyright}, RISHAD A. IRANI\textsuperscript{\textcopyright}, (Member, IEEE), AND MOJTABA AHMADI\textsuperscript{\textcopyright}, (Senior Member, IEEE)

Department Mechanical and Aerospace Engineering, Carleton University, Ottawa, ON K1S 5B6, Canada

Corresponding author: Steffan Lloyd (steffan.lloyd@carleton.ca)

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant RGPIN-2017-06967, Grant RGPIN-2015-04169, and Grant CRDPJ 514258-17. The work of Steffan Lloyd was supported in part by the Ontario Graduate Scholarship (OGS) and in part by the NSERC CGS-M Scholarship.

\textbf{ABSTRACT} We present a novel numerical integration technique, Neural Network Integration, or NNI, where shallow neural network design is used to approximate an integrand function within a bounded set. This function approximation is such that a closed-form solution exists to its definite integral across any generalized polyhedron within the network’s domain. This closed-form solution allows for fast integral evaluation of the function across different bounds, following the initial training of the network. In other words, it becomes possible to “pre-compute” the numerical integration problem, allowing for rapid evaluation later. Experimental tests are performed using the Genz integration test functions. These experiments show NNI to be a viable integration method, working best on predictable integrand functions, but worse results on singular and non-smooth functions. NNI is proposed as a solution to problems where numerical integrations of higher dimension must be performed over different domains frequently or rapidly and with low memory requirements, such as in real-time or embedded engineering applications. The application of this method to the optimization of integral functions is also discussed.

\textbf{INDEX TERMS} Numerical integration, neural network, optimization.

I. INTRODUCTION

The need to integrate across multiple dimensions frequently arises in many engineering, statistics, and financial problems [1]. In many cases, the integration does not have a closed-form solution, meaning that approximate numerical methods must be used instead. Currently, multidimensional numerical integration can be done through a variety of methods. The classical one-dimensional approaches of trapezoidal rule or Simpson’s rule can be expanded to $N$ dimensions through “product rules,” by taking Cartesian products of each dimension [1], [2]. Unfortunately, the error bound of these methods, as well as other piecewise quadrature methods, degrades with increasing dimension. This degradation is known as the “curse of dimensionality,” whereby the computing cost grows exponentially with dimension [3]. Sparse grid methods offer slightly better error convergence, but still struggle in high dimensions, and only work for highly regular functions [4].

The Monte Carlo integration method can perform well in very high dimensional space, as its complexity is independent of the dimension of the problem [3]. However, it is a stochastic method and thus only provides a probabilistic bound on integration error. Furthermore, the error convergence of the Monte Carlo method is slow, as it does not benefit from any regularity the integrand function may offer. Quasi-Monte Carlo integration methods use well-chosen deterministic grids to ensure uniform spacing of the sample points, and can achieve deterministic error bounds. These methods can be shown to have slightly better error convergence [1], but still not comparable to that of quadrature techniques. Adaptive methods sample the integrand function iteratively in an attempt to gain the most information about the function in the regions where the highest variance occurs. These methods can be applied to both quadrature methods [5], [6] and quasi-Monte Carlo methods [7], [8].

A common trait of all the above methods is that there is little that can be done to “pre-compute” the integration problem. Accordingly, in applications where integrations of a function must be carried out many times over different
regions, the computations involve repetitive calculations at high cost. If severe restrictions are placed on the computation power available, such as in real-time applications, then the problem can become intractable. This issue is further compounded if the integrand function itself is numerically expensive to compute.

This paper introduces Neural Network Integration (NNI), a novel numerical integration methodology, as a means of solving these types of problems in a tractable way. NNI is a numerical integration algorithm which uses a shallow neural network as the interpolation function. The main advantage of NNI is that it allows the bulk of computational processing to be performed a priori, allowing for rapid computation later. In NNI integration, a shallow neural network model is created to approximate the integrand function within a bounded integration region. Neural networks are convenient means of representing functions since they are universal approximators [9] and generalize well into multiple dimensions when compared to splines [10]. While network training is computationally expensive, it only needs to be done once and can be performed on specialized hardware. Once training is complete, however, the network design is such that a closed-form solution to the integral exists across the network inputs. As such, subsequent evaluations of the integral for different sets of bounds can be computed rapidly.

The characteristics of NNI make it ideal for problems where repeated integrations of the same function must be performed across different bounds with high accuracy, little processing time, and low memory requirements. These characteristics occur in many engineering problems, in particular in real-time or embedded applications where computation power is limited or where lookup-table style solutions are precluded due to a lack of memory. We also show the application of NNI to the optimization of integral functions, where many integral evaluations are required. In optimization, it will be shown that NNI has the added advantage of allowing for easy estimation of the optimization gradient, facilitating the use of efficient gradient-based methods. A possible application of NNI in optimization is in the pose optimization of a robotic manipulator, where it is necessary to integrate over the six-dimensional space of positions and orientations of the end-effector in order to place the robot’s task in an optimal region of its workspace.

The idea of using shallow neural network function approximators in numerical integration has been researched briefly before. Zhao and Hut [11] outlined a method for numerical integration with a cosine-based neural-network architecture. However, the cosine activation function does not allow for the approximation of all classes of functions and can be difficult to train due to being non-monotonic [12]. Furthermore, the approach only allowed for 1-dimensional function integration, which is more quickly accomplished through other quadrature methods. The proposed NNI method uses a much more flexible, adaptable, and trainable network architecture, which still allows for analytical integration. Furthermore, NNI can be applied to an arbitrary level of dimensionality, where the integration problems become less tractable, and the need for pre-computation is greater.

The organization of this paper is as follows. Section II formally presents the problem and the proposed Neural Network Integration theory, methods, and implementational notes. Section III presents numerical testing results for the proposed method in terms of both accuracy and evaluation time. Finally, Section IV discusses possible applications of the method, in particular in the solution to optimization problems of high-dimensional integral functions.

II. NEURAL NETWORK INTEGRATION (NNI)
Let $\mathcal{O}$ be a compact subset of $\mathbb{R}^n$, over which a continuous real function $f : \mathbb{R}^n \to \mathbb{R}$ is defined. Let $\mathcal{S}$ be a subset of $\mathcal{O}$. The problem investigated in this paper is to evaluate the numerical integral of $f$ across $\mathcal{S}$ in the form

$$I(f) = \int_{\mathcal{S}} f(x) \, dx. \quad (1)$$

This paper will consider only the case where $\mathcal{S}$ is a convex, bounded set in $\mathcal{O}$ which is characterized by a finite number of linear inequalities or equalities. Thus, $\mathcal{S}$ can be defined compactly as

$$\mathcal{S} = \{ x \in \mathcal{O} \mid A_1 x \preceq b_1, A_2 x = b_2 \} \quad (2)$$

where $\preceq$ denotes a component-wise vector inequality. $A_1 x \preceq b_1$ denotes the intersection of a set of halfspaces, and $A_2 x = b_2$ denotes the intersection of a set of hyperplanes. Geometrically, (2) is the formula for a generalized polyhedron of dimension $d$ in $\mathbb{R}^d$, where $d = n - \text{rank}(A_2)$. This definition includes the common case of hypercube integration in $\mathbb{R}^d$, but can also be extended to much more complex domains.

![Figure 1. The proposed function approximation network structure.](image)

This paper proposes solving (1) by creating a function estimator for $f(x)$, denoted by $\hat{f}(x)$, using an artificial neural network architecture. Figure 1 shows a graphical representation of the proposed neural network function $\hat{f}(x)$ with $n$ inputs, a single hidden layer with $k$ nodes, and a single
summing output node. Mathematically, \( \hat{f}(x) \) has the form

\[
\hat{f}(x) = b^{(2)} + \sum_{j=1}^{k} w_j^{(2)} \phi \left( b_j^{(1)} + \sum_{i=1}^{n} w_{ji}^{(1)} x_i \right)
\]

where

- \( w_j^{(1)} \) and \( w_j^{(2)} \) are the elements of \( W^{(1)} \in \mathbb{R}^{k \times n} \) and \( W^{(2)} \in \mathbb{R}^{k \times 1} \), and represent the weights of the first and second layers, respectively,
- \( b_j^{(1)} \) and \( b_j^{(2)} \) are the elements of \( b^{(1)} \in \mathbb{R}^k \) and \( b^{(2)} \in \mathbb{R} \) and represent the biases of the first and second layers, respectively,
- \( x_i \) are the elements of \( x \in \mathbb{R}^n \), and
- \( \phi \) is the logistic sigmoid function, defined as

\[
\phi(z) = \frac{1}{1 + e^{-z}}.
\]

To show the feasibility of integrating \( \hat{f}(x) \) in the place of \( f(x) \), we show two results. First, it will be shown that it is possible to make the integration error \( |\hat{I}(f) - I(f)| \) arbitrarily small, where

\[
\hat{I}(f) = \int_S \hat{f}(x) \, dx.
\]

Secondly, it is demonstrated that a closed-form formulation of the definite integral \( I(f) \) over an \( S \subseteq \mathcal{O} \) is possible, and can be computed rapidly. Finally, examples will be given to show how these theorems can be applied practically.

Theorem 1 (Universal Approximation Theorem [9], [13], [14]): For a given \( f(x) \) and any \( \epsilon > 0 \), one can find a network size \( k \) and weights \( W^{(1)}, W^{(2)}, b^{(1)} \) and \( b^{(2)} \), such that the approximation error of \( \hat{f}(x) \) has an upper bound

\[
|\hat{f}(x) - f(x)| < \epsilon
\]

for all \( x \in \mathcal{O} \).

The proof of this result is covered in detail in the cited references. The only observation required is that the proposed architecture of \( \hat{f}(x) \) is congruent with the proofs therein, namely that

1) the chosen logistic activation function is bounded, continuous and non-constant;
2) the integrand function \( f(x) \) is continuous; and
3) the approximation domain \( S \) is bounded and compact.

Corollary 1: For a given \( f(x) \) and any \( \xi > 0 \), one can find a network size \( k \) and weights \( W^{(1)}, W^{(2)}, b^{(1)} \) and \( b^{(2)} \) such that the integration error of \( \hat{f}(x) \) over \( S \) has an upper bound

\[
|\hat{I}(f) - I(f)| < \xi
\]

for any finite \( S \subseteq \mathcal{O} \).

Proof: From Theorem 1, \( \hat{f}(x) \) can be chosen such that there exists an upper bound \( \epsilon \) to the absolute error of the network for all \( x \in \mathcal{O} \). Then, it is simple to show that the error on \( \hat{I}(f) \) is also bounded,

\[
|\hat{I}(f) - I(f)| = \left| \int_S \hat{f}(x) \, dx - \int_S f(x) \, dx \right| = \int_S \left| \hat{f}(x) - f(x) \right| \, dx
\]

where \( V(S) \) is the n-dimensional volume of \( S \). \( S \) has finite volume, as it is a subset of the compact set \( \mathcal{O} \). It follows that the number \( \epsilon \) \( V(S) \) is also finite. Accordingly, there exists a network configuration such that the approximation error is bounded by \( \xi = \epsilon V(S) \), if the neural network is selected such that \( \epsilon \leq \xi / V(S) \).

If a network \( \hat{f}(x) \) is fully trained to approximate \( f(x) \) over \( \mathcal{O} \), then network parameters concisely contain the entirety of the training data, and no further evaluations of the original function \( f(x) \) need to be performed. In cases where this integrand function itself is computationally expensive, \( \hat{f}(x) \) already provides a convenient way of computing the function value quickly within \( \mathcal{O} \), or interpolating between the existing function evaluations.

To establish the theory of NNI, we first note that the logistic sigmoid function \( \phi \), used in the neural network architecture, is of the same form as \( \text{Li}_0 \), the polylogarithm (or Jonquières’ function) of order 0,

\[
\text{Li}_0(z) = -\frac{1}{1 - e^{-z}}.
\]

As such, \( \phi \) can be written, for an arbitrary argument \( \kappa(z) \), as

\[
\phi(\kappa(z)) = \frac{1}{1 + e^{-\kappa(z)}} = -\text{Li}_0\left(-e^{\kappa(z)}\right).
\]

\[
\text{Lemma 1: If } \kappa(z) \text{ is linear in } z, \text{ then a closed-form solution to the antiderivative of } -\text{Li}_0\left(-e^{\kappa(z)}\right), s \in \mathbb{Z}, \text{ is possible in terms of polylogarithm functions.}
\]

Proof: It can be easily verified that the antiderivative of \( -\text{Li}_0\left(-e^{\kappa(z)}\right) \) is of the form

\[
\int -\text{Li}_0\left(-e^{\kappa(z)}\right) \, dz = -\left( \frac{\partial \kappa(z)}{\partial z} \right)^{-1} \text{Li}_{s+1}\left(-e^{\kappa(z)}\right),
\]

wherehere integration constants are ignored as the current work involves definite integrals only. This formulation is possible so long as \( \partial \kappa(z)/\partial z \) is a constant, or, equivalently, that \( \kappa(z) \) is linear in \( z \).

\[
\text{Lemma 2: If } \kappa(z_1, \ldots, z_n) \text{ is linear in } z_1, \ldots, z_n, \text{ and } \phi(\kappa(z_1, \ldots, z_n)) \text{ is integrated in } n \text{ dimensions over } z_1, \ldots, z_n \text{ between some set of limits } a_1, \ldots, a_n, \text{ and } b_1, \ldots, b_n, \text{ then a closed-form solution to this integral in terms of polylogarithm functions is possible if the integration limits are also linear in } z_1, \ldots, z_n.
\]
Proof: From (10), this statement is analogous to
\[
\int_{\alpha_n}^{\beta_n} \int_{\alpha_{n-1}}^{\beta_{n-1}} \cdots \int_{\alpha_1}^{\beta_1} f(x) dx_1 \cdots dx_n
\]
where all but the final integration limits \(\alpha_n\) and \(\beta_n\) can be a function of the remaining integration variables. By Lemma 1, each integration in \(z_i\) of this formulation will be possible so long as the argument of the exponential is linear in \(z_i\). Since it is assumed that \(k(z_1, \ldots, z_n)\) is linear in \(z_1, \ldots, z_n\), then all integrations will be possible so long as the substitutions of the integration limits into \(k(z_1, \ldots, z_n)\) do not break this linearity. This condition is possible if each of the integration limits themselves, \(\alpha_i(z_{i+1}, \ldots, z_n), \beta_i(z_{i+1}, \ldots, z_n), i = 1, \ldots, n\) are linear in \(z_1, \ldots, z_n\).

Theorem 2 (Neural Network Integration): If a single-layer neural network of the form of \(f(x)\) is integrated over a closed generalized polyhedron in dimension \(n\),
\[
\tilde{S} = \{x \in \mathcal{O} \mid \mathbf{A}_1 \mathbf{x} \leq \mathbf{b}_1\},
\]
then a closed-form solution is possible in terms of polylogarithm functions.

Proof: We note that the network forward propagation function \(f(x)\) is a linear combination of the logistic sigmoid function and constant terms. Furthermore, the argument of the sigmoid function in (3) is linear in all the input variables. Therefore, it is possible to apply Lemma 2 so long as the integration limits are linear, which is enforced by the predicate \(\mathbf{A}_1 \mathbf{x} \leq \mathbf{b}_1\) on \(\tilde{S}\). As such, a closed-form solution to the \(n\)-th order integral across the inputs of the network will exist and can be determined in closed-form.

By Theorems 1 and 2, any continuous real function \(f\) can be approximated arbitrarily well by a single-layer neural network with a logistic sigmoid activation function, and such a network can be analytically integrated across any domain as in (13). It is also possible that Theorem 2 could be extended to include similar results for other often-used activation functions. Such work would an interesting area for future work in the field.

The statement of Theorem 2 is admittedly quite general and does not provide any guidance on how such a closed-form solution may be found. We now provide three examples to illustrate NNI’s application to some common cases.

Example 1 (1-Dimensional Integration): Let \(I(f)\) be 1-dimensional integral of the form
\[
I(f) = \int_{\alpha_1}^{\beta_1} f(x) dx,
\]
through substitution with (10) and (11), the approximated integral \(\hat{I}(f)\) of the neural network function is
\[
\hat{I}(f) = b^{(2)}(\beta_1 - \alpha_1) + \sum_{j=1}^{k} w_j^{(2)} \left[ (\beta_1 - \alpha_1) + \frac{\Phi_j}{w_{ij}^{(1)}} \right],
\]
with \(\Phi_j\) as the sigmoid integral term of the \(j\)-th neuron, defined by
\[
\Phi_j = \frac{\mathbf{w}_1}{\mathbf{w}_2} - \sum_{i=1}^{n} w_i^{(1)} (\beta_1 - \alpha_1) + \frac{\Phi_j}{w_{ij}^{(1)}}.
\]

Example 2 (Rectangular Integration): Example 1 can be extended to the 2-dimensional case as
\[
I(f) = \int_{\alpha_1}^{\beta_1} \int_{\alpha_2}^{\beta_2} f(x) dx_1 dx_2.
\]

The corresponding approximated integral can be shown to be

Through substitution with (10) and (11), the approximated integral \(\hat{I}(f)\) of the neural network function is
\[
\hat{I}(f) = b^{(2)}(\beta_1 - \alpha_1)(\beta_2 - \alpha_2) + \sum_{j=1}^{k} w_j^{(2)} \left[ (\beta_1 - \alpha_1)(\beta_2 - \alpha_2) + \frac{\Phi_j}{w_{ij}^{(1)}} \right].
\]

Example 3 (n-Dimensional Hyperrectangle): Take the case where \(\tilde{S}\) is an \(n\)-dimensional hyperrectangle in \(\mathbb{R}^n\) \([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \cdots \times [\alpha_n, \beta_n]\) with \(I(f)\) defined as in (1). Then, the definite integral of this function across all inputs can be calculated as
\[
\hat{I}(f) = b^{(2)} \prod_{i=1}^{n} (\beta_i - \alpha_i) + \sum_{j=1}^{k} w_j^{(2)} \left[ \prod_{i=1}^{n} (\beta_i - \alpha_i) + \frac{\Phi_j}{\prod_{i=1}^{n} w_{ij}^{(1)}} \right],
\]
where
\[
\Phi_j = \sum_{r=1}^{2^n} \xi_r, \mathbf{L}_r \left[ -\exp \left[ -b_j^{(1)} - \sum_{i=1}^{n} w_{ij}^{(1)} \xi_{i,r} \right] \right].
\]
\(\xi_r\) is the sign of the \(r^{th}\) sigmoid integration term of the node, and \(\xi_r, r\) is the appropriate integration limit for the \(r^{th}\) dimension and \(r^{th}\) summation element. The quantities \(\xi_r\) and \(\xi_r, r\) can be computed directly using the formulae

\[
\xi_r = \prod_{d=1}^{n} (-1)^{r/2^{n-d}}, \quad (18c)
\]

Since the substitutions into \(\hat{f}\) are linear, the linearity of the arguments in the integrand is not broken. Furthermore, because the constraints defined by \(h\) are affine, the partial derivative terms will be constants. Thus, the norm of the vector they define will also be a constant. The integration of \(\hat{f}(x)\) is then simply scaled by a real number, and still in the same form as is required for Theorem 2.

Corollary 3 (NNI With Change of Variables): Let \(\hat{f}(x)\) be integrated over a closed \(d\)-dimensional generalized polyhedron \(S\in\mathbb{R}^n\). If \(g : S \rightarrow S' \in \mathbb{R}^d\) is an affine, continuous and one-to-one function, then an analytical formulation of this integral is possible after a change of variables defined by \(g\).

Proof: If \(\hat{I}(f)\) is integrated using an affine change of coordinates \(g : S \rightarrow S' \in \mathbb{R}^d\), such that \(x = g(y)\), then the integration in \(\hat{I}(f)\) becomes

\[
\hat{I}(f) = \int_{S'} \hat{f}(g(y)) |\det(J_g)| dy,
\]

where \(\det(J_g)\) is the determinant of the Jacobian of the transformation, defined as

\[
\det(J_g) = \det\begin{bmatrix}
\frac{\partial g_1}{\partial y_1} & \cdots & \frac{\partial g_1}{\partial y_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial g_n}{\partial y_1} & \cdots & \frac{\partial g_n}{\partial y_n}
\end{bmatrix}.
\]

Theorem 2 requires the argument to the sigmoid function in (3) to be linear in the integration variable. If \(g\) is affine, then the mapping will not break the linearity of this argument. Additionally, the linearity of \(g\) implies that \(\det(J_g)\) is a constant, and that if \(S\) satisfies (2), then \(S'\) will too. Finally, since \(g\) is one-to-one and class \(C_1\), the Jacobian exists and is non-singular. It follows then \(\det(J_g)\) is defined and constant. Then, the integration is still in the same form as is required for Theorem 2.

These corollaries show that NNI can be applied to more complex domains than the hyperrectangles of Example 3. To illustrate how they may be used, we provide a further example in which Corollaries 2 and 3 are both necessary.

Example 4 (d-Dimensional Rotated Hyperrectangle): Consider the case where \(S\) is a \(d\)-dimensional hyperrectangle in \(\mathbb{R}^n\), defined in the hyperplane \(A_2 x = b_2\), and constrained by a set of \(d\) upper and lower integration bounds, which have been rotated by an arbitrary orthonormal rotation matrix \(R \in \mathbb{R}^{d \times d}\). For this example, we define the integration limits as acting on the first \(d\) dimensions of \(O\). In other words, we seek to integrate over the hyperrectangle which lies in the hyperplane \(A_2 x = b_2\) and normal to the cubic region defined by \([\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \cdots \times [\alpha_d, \beta_d]\) after a rotation \(R\). A visualization of this domain with \(n = 3\) is given in Figure 2, where \(O = \mathbb{R}^3, S \subset \mathbb{R}^2\) is defined on the plane defined by a single linear equality constraint \(A_2 x = b_2\), and \(R\) is a counter-clockwise rotation about the \(z\)-axis.

Define the free variables \(x_f = [x_1, \ldots, x_d]^T\) be the first \(d\) elements of \(x\), and let \(A_{2f}, A_{2c}\) be defined as in (20). Then we
can write
\[
\begin{align*}
x &= \begin{bmatrix} x_f \\ A_{2x}^\dagger (b_2 - A_{2x} x_f) \end{bmatrix} = \begin{bmatrix} x_f \\ C x_f + d \end{bmatrix} \\
&= \begin{bmatrix} x_f \\ C x_f + d \end{bmatrix} (24)
\end{align*}
\]
where \( C = -A_{2x}^\dagger A_{2x}, d = A_{2x}^\dagger b_2 \). With \( \alpha = [\alpha_1, \ldots, \alpha_d]^T \) and \( \beta = [\beta_1, \ldots, \beta_d]^T \), \( S \) can then be written as
\[
S = \left\{ x \in \mathcal{O} \mid \begin{bmatrix} -R \\ R \end{bmatrix} d_{x,m} x \leq \begin{bmatrix} -\alpha \\ \beta \end{bmatrix}, A_{2x} = b_2 \right\}.
\]
(25)
The integral \( I(f) \) is then rewritten using (21) from Corollary 2 as
\[
I(f) = \int_S f(x) \, dx
= \int_S f \left( \begin{bmatrix} x_f \\ C x_f + d \end{bmatrix} \right) \prod_{i=1}^{n-d} \left\| C_{li} \cdots C_{di} 1 \right\| \, dx
= K \int_S \left( \begin{bmatrix} x_f \\ C x_f + d \end{bmatrix} \right) \, dx
\]
(26)
where the constant \( K = \prod_{i=1}^{n-d} \left\| C_{li} \cdots C_{di} 1 \right\| \). Finally, the integration bounds can be explicitly written by a change of variables \( y_f = R x_f \) using (22) from Corollary 3 as
\[
I(f) = K \left| \det (R) \right| \left( \begin{bmatrix} y_f \\ C y_f + d \end{bmatrix} \right) dy_{f1} \cdots dy_{fd}
= K \left( \begin{bmatrix} y_f \\ C y_f + d \end{bmatrix} \right) dx_1 \cdots dx_d,
\]
(27)
where \( |\det (R)| = 1 \) since \( R \) is orthonormal. Since \( I(f) \) in (27) is of the same form as in Example 3 with the substitution \( x = \begin{bmatrix} (R x_f)^T, (C R x_f + d)^T \end{bmatrix} \), the integral \( I(f) \) can be computed as directly from (18) by replacing \( \alpha_i \) and \( \beta_i \) with the elements of the \( n \)-vectors \( \alpha' \) and \( \beta' \) respectively, as
\[
\begin{align*}
\alpha' &= \left( (R x_f)^T, (C R x_f + d)^T \right)^T,
\beta' &= \left( (R \beta)^T, (C R \beta + d)^T \right)^T,
\end{align*}
\]
(28)
then finally scaling the result by the constant \( K \).

A. IMPLEMENTATION NOTES

We will now discuss the implementational concerns of how NNI can be practically used. The main difficulties arise in the sizing and training of the neural network approximation itself. These points are addressed briefly in the following sections.

1) FUNCTION SAMPLING

How the function is sampled affects the performance of NNI, as with any point evaluation based numerical integration routine. Areas of the function with high variance cannot be well approximated if they are not appropriately sampled. The current study on NNI uses standard uniform grid sampling of the function for all experiments. However, it is proposed that future work in NNI should investigate variance reduction techniques and adaptive sampling methods as a means of potentially increasing the accuracy of the NNI.

2) NETWORK SIZING

Network sizing is a reoccurring question in neural network design, and correct sizing is crucial to allow the network to learn and generalize properly without overfitting the data. There is no exact answer to the network sizing question [18], and though many heuristics have been proposed, none are universally accepted. The ideal number of nodes \( k \) depends on the complexity of the function being evaluated, and the number of function samples available. If the network is sized too large but trained with very few points, the resulting network will likely have high variance and overfit the data, leading to poor interpolation between the sampled points, and corresponding poor integration results. Conversely, if the network size is too small, the network will be unable to capture the complexity of the function and also give poor results.

Analytical results do exist; however, they are of little help. With the logistic sigmoid activation functions in the proposed network, the number of nodes required for negligable error on a training set of \( N \) samples can be as high as \( N \) [19]. Baum and Haussler (2008) proposed that the number of weights should be around ten times smaller than the number of training samples [20]. For the proposed network structure, this guideline would suggest approximately \( k = \lfloor 0.1 N/(n+2) \rfloor \). However, in testing, this formula was found to give poor interpolatory performance in NNI, as \( k \) was often too high – particularly with large \( N \). In the current study, the following heuristic is proposed for determining the network hidden layer size,
\[
k = \left[ \left( \log_{10}(N) \right)^{-k_1} f_2 N \left( n + 2 \right) \right].
\]
(30)
where \(K_1\) and \(K_2\) are tunable parameters. This formula is a modification to the guideline of Baum and Haussler, which adds a decaying logarithmic term that prevents \(k\) from growing too large at larger values of \(N\). This modified form was seen to give good general performance; however we offer no analytical justification for its use.

3) NETWORK TRAINING
Choosing the weights and biases such that the approximation is sufficiently accurate over \(O\) from sample points is a supervised learning problem. Network training is a subject that has been extensively covered in literature [10], [21]–[24]. The training of such a network is generally done using gradient-descent methods or variations thereof. In the current study, network evaluation is performed using the mean-squared error metric, and network training is performed using the Levenberg-Marquardt backpropagation algorithm [23]. This algorithm is found to be one of the fastest training methods for small to medium-sized networks, and generally achieves the highest convergence for a given training period [25], [26].

4) NETWORK NORMALIZATION
One common form of preprocessing that was found to be necessary for NNI was a linear scaling of the input and output variables into the same range as the output of the activation functions. This mapping helps reduce the training time of gradient descent methods, and also helps to keep the range of the activation function inputs near the origin. This scaling is particularly important in NNI since, during integration, inputs to the function become the arguments of the polylogarithm functions. In general, the polylogarithm function is not bounded, and evaluating it far from the origin can introduce numerical imprecision into the results.

The strategy used in this paper is simple linear min-max mapping of the input and output of the function into the range of \([y_{\text{min}}, y_{\text{max}}]\) using

\[
x'_i = \frac{y_{\text{max}} - y_{\text{min}}}{x_{i,\text{max}} - x_{i,\text{min}}} (x_i - x_{i,\text{min}}) + y_{\text{min}},
\]

\[
f'(x) = \frac{y_{\text{max}} - y_{\text{min}}}{f_{\text{max}} - f_{\text{min}}} (f(x) - f_{\text{min}}) + y_{\text{min}},
\]

where \(x_{i,\text{min}}\) and \(x_{i,\text{max}}\) are the minimum and maximum respectively of the \(i\)th input \(x_i\), and \(f_{\text{min}}\) and \(f_{\text{max}}\) are the minimum and maximum of the sampled function data. The network is then trained to fit the function which maps the scaled inputs to the scaled output.

When the normalized network is integrated, some additional details must be taken into account. First, the integration limits must also be scaled using the same factors as the inputs in (31). Then, a scaled integral \(\hat{I}(f)\) can be computed from (18). The true integral is obtained by rescaling \(\hat{I}(f)\), as

\[
\hat{I}(f) = \frac{V(S)(f_{\text{max}} - f_{\text{min}})}{V(S')(y_{\text{max}} - y_{\text{min}})} \hat{I}(f) + \left( f_{\text{min}} - \frac{f_{\text{max}} - f_{\text{min}}}{y_{\text{max}} - y_{\text{min}}} y_{\text{min}} \right) V(S),
\]

where \(V(S)\) and \(V(S')\) are the original and scaled hypervolume of the integration, respectively. In the case of a hyperrectangle integration as in Example 3, these volumes are computed as

\[
V(S) = \prod_{i=1}^{n} (x_{i,\text{max}} - x_{i,\text{min}}),
\]

\[
V(S') = (y_{\text{max}} - y_{\text{min}})^n.
\]

III. TEST RESULTS
This section demonstrates the feasibility of NNI using experimental validation on test integrals. We stress that these tests are done to show the feasibility, not the superiority, of NNI in terms of accuracy. The advantages of NNI lie in its ability to produce results with an acceptable level of accuracy, and reduced execution time and memory.

Testing is done using the numerical integration testing package by Genz [27], [28], and further refined by Joe and Sloan [2]. The package consists of six families of integrands, \(f^{(1)}, \ldots, f^{(6)}\), defined over the unit cube \([0, 1]^n\):

**Oscillatory:**

\[
f^{(1)}(x) = \cos \left( 2\pi u_1 + \sum_{i=1}^{n} c_i x_i \right),
\]

**Product Peak:**

\[
f^{(2)}(x) = \prod_{i=1}^{n} \left[ c_i^2 + (x_i - u_i)^2 \right],
\]

**Corner Peak:**

\[
f^{(3)}(x) = \left( 1 + \sum_{i=1}^{n} c_i x_i \right)^{-(n+1)},
\]

**Gaussian:**

\[
f^{(4)}(x) = \exp \left( - \sum_{i=1}^{n} c_i^2 (x_i - u_i)^2 \right),
\]

**Continuous:**

\[
f^{(5)}(x) = \exp \left( - \sum_{i=1}^{n} c_i |x_i - u_i| \right),
\]

**Discontinuous:**

\[
f^{(6)}(x) = \begin{cases} 0, & \text{if any } x_i > u_i, \\ \exp \left( \sum_{i=1}^{n} c_i x_i \right), & \text{otherwise.} \end{cases}
\]

Each function is parameterized by two \(n\)-vectors, \(u\) and \(c\). The variable \(u\) is a shift parameter with \(u_i\) randomly distributed in \([0, 1]\) and does not significantly affect the difficulty of integration. The variable \(c\) controls the difficulty of integration and is determined for each integrand family \(f^{(j)}\) as

\[
c_j = \left( \frac{h_j}{n^d \sum_{i=1}^{n} c_i} \right)^{c'},
\]

where \(c'\) is an \(n\)-vector with its components randomly distributed in \([0, 1]\), and the values \(h_j, e_j\) are fixed for each integrand family and dimension [2], [27], [28].

We use this testing package to quantify the integration accuracy and evaluation time of NNI, and provide three algorithms for comparison. Testing of NNI is performed using uniform sampling distribution. The number of hidden nodes \(k\)
is selected as per (30), with \( K_1 = 4.33 \) and \( K_2 = 16 \). Network training is done using 90% of the available points, and the remaining 10% of points are used for network validation. In cases where the validation MSE of the network is more than two orders of magnitudes larger than the training MSE, the network is considered overfit and is re-trained. Network inputs and outputs are mapped into the range \([-1, 1]\) using linear min-max mapping, as in Section II-A.4. Network training is performed for 5000 epochs. Once training is completed, the estimated integral is evaluated directly using (18).

Two non-adaptive sampling algorithms are provided for comparison: TRAPZ and MC. TRAPZ is an \( n \)-dimensional product-rule implementation of the standard trapezoidal rule (linear interpolation). MC is a Monte Carlo integration routine, based on random point sampling. Additionally, we provide an adaptive sampling algorithm ADAPT for comparison. ADAPT is an adaptive quadrature method developed by Van Dooren and De Ridder [6] with modifications by Genz and Malik [5]. However, we emphasize that ADAPT here has a significant advantage over the uniform sampling methods, NNI included, as it is able to gather more information from areas of high variance in the function. We provide it as a comparison and note that NNI, and the other uniform sampling methods, could be similarly extended to use adaptive sampling methods and also expect improved performance.

### A. INTEGRATION ACCURACY

Testing was performed for each integrand family (36a-i), in dimension 2 and 6, for logarithmically increasing allowable number of integrand calls \( N \). The results are shown for each integrand family in Figure 3. In each plot, the line-type denotes the integration method used, and the marker type denotes the dimension of integration. We take the number

\[
\log_{10} \left| \frac{I(f)}{I(f) - I(f)} \right| = \frac{1}{N} \sum_{j=1}^{N} \log_{10} \left| \frac{I(f)}{I(f) - I(f)} \right|
\]

(38)

to be the measure of performance, which can be interpreted as the correct number of digits of the integral estimate. For each point, a sample of 20 evaluations was taken, and the set median was shown on the figure. The integrand parameters \( e_j \) and \( h_j \) used are listed in Table 1. These parameters were selected similarly to Sloan [2], with the goal of keeping the integration difficulty of each family approximately equal.

Note that both TRAPZ and ADAPT require specific structure to their sampling points, and as such, the number of integration calls in these methods was sometimes lower than the number of integrand calls \( N \). For these algorithms, \( N \) represents the maximum permissible integrand calls.

The testing shows promising results. NNI, in general, had good accuracy within the non-adaptive algorithms, and performed best on functions with higher regularity, and in the tests at higher dimensions. The ADAPT algorithm, in most cases, had the best overall performance. In particular, in the function families with large local variability, the adaptive algorithm is expected to perform much better than uniform sampling methods, as it can gain more information in areas where the function changes more. This advantage is particularly visible with the integrand families which had large local complexity, such as the \( f^{(3)} \) : corner peak and \( f^{(4)} \) : Gaussian families, or non-regularities, such as the \( f^{(5)} \) : continuous or \( f^{(6)} \) : discontinuous families. The non-adaptive algorithms do not have sufficient information in the high-variance regions to interpolate accurately.

Conversely, in the more predictable families such as \( f^{(1)} \) : oscillatory and \( f^{(2)} \) : product peak, we see that NNI performs very well, even when compared to ADAPT. This result is logical since these functions have a more constant regularity to them, which means that adaptive sampling provides little benefit. In particular, the oscillatory family offers almost no benefit to adaptive routines since the variance is constant everywhere. In some cases for this family, we see that NNI in fact outperforms the adaptive routine. These results lead us to believe that extensions to the current study, such as integrating NNI with a variance-based adaptive sampling algorithm, or intelligently weighting the error terms of the neural network during training, could both lead to higher accuracy results on integrand functions with discontinuities, singularities, or otherwise varying regularity.

It is also noted that the performance of TRAPZ and ADAPT (relative to the other methods) is significantly better for \( n = 2 \) as opposed to \( n = 6 \). ADAPT still performs relatively well at \( n = 6 \), although generally gave results approximately equal or even worse than the uniformly-sampled NNI. TRAPZ at this dimension performed very poorly, and gave less accurate results than even the non-interpolatory MC algorithm. This trend of deteriorating performance with increasing dimension is expected of product-based methods and should continue as dimensionality increases. Such methods become completely intractable as dimension \( n \) becomes even moderately large (\( n = 10 - 20 \)) [2]. MC, being a Monte Carlo method, is entirely dimension-independent, and its performance depends only on the difficulty of the function being integrated [1]. NNI appears to also be somewhat agnostic to dimension, performing relatively as well as MC in different dimensionalities. This behavior is expected of NNI, as neural networks, in general, are known to extend well into large dimensionalities [10]. The ability of NNI to interpolate well in high dimensions with a small number of sample points is

### Table 1. Difficulty parameters used for integration testing.

| Integrand Family | \( f^{(1)} \) | \( f^{(2)} \) | \( f^{(3)} \) | \( f^{(4)} \) | \( f^{(5)} \) | \( f^{(6)} \) |
|------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| **Dim 2**        |             |             |             |             |             |             |
| Difficulty \( h_j \) | 100 | 150 | 600 | 150 | 100 | 16 |
| Exponent \( e_j \) | 1 | 1 | 1 | 1 | 1 | 1 |
| **Dim 6**        |             |             |             |             |             |             |
| Difficulty \( h_j \) | 300 | 300 | 1000 | 250 | 200 | 48 |
| Exponent \( e_j \) | 1.75 | 1.75 | 1.75 | 1.75 | 1.75 | 1.75 |
FIGURE 3. Test results for the 6 Genz families of integrands.

**NNI points shown with solid marker fill (• and ▲) use a smaller sample size of 5, due to the longer training times at large $N$, and the computation restrictions of the study.**

†Plots are similarly scaled to allow for comparison. However, some data is clipped as a result.

‡The discontinuous family is particularly difficult to integrate, resulting in some stochastic results with lower $N$. Larger sample sizes would be required to eliminate this behavior, but are not possible due to computational restrictions of the current study.
a significant strength when compared to the product methods tested.

We also point out that NNI can perform in a broader set of conditions than other quadrature algorithms, as it does not require the function sample points to fit any specific pattern. This flexibility contrasts the ADAPT and TRAPZ algorithms, as well as many other quadrature algorithms, which require integrand evaluations at pre-specified points in the integration region. This characteristic suggests that NNI can be particularly suitable when the integrand function can only be sampled sparsely and at arbitrary points (such as with data collected experimentally). However, the performance of NNI in these cases would be dependent on proper network sizing and variance-bias balancing; thus, we do not generalize further on this point.

The discontinuous family of functions is of particular difficulty to all methods. We note that this family is not of class $C^0$, and as such, does not fall into the allowable form for Theorem 1, so NNI is not guaranteed to give good results on such functions. ADAPT also has difficulties with this family, as evinced by its somewhat stochastic performance at larger $N$.

Finally, the reader may note a general decrease in NNI performance at higher point levels, in some cases. Unfortunately, this trend is an artifact of the computational limits of our study. Large point sets used a larger network size, which did not train sufficiently within the allocated 5000 epochs. However, especially for larger $N$, we noted a strong correlation between training time and integration performance. In practice, NNI training time would not need to be limited in the same way (see Section IV), and training could be done over long periods on specialized hardware.

B. INTEGRATION TIME

Additional testing was done to characterize the integration time of the compared methods. For this test, we use only the oscillatory integrand family $f^{(1)}$. Ten samples were used for each algorithm (NNI, TRAPZ, MC and ADAPT), for dimensions 2 and 6, for increasing values of $N$. In this test, the network size used in NNI is kept constant at $k = 100$. Testing was done using MATLAB interpreted code on a Linux system, 128 GB RAM, 1.4 GHz CPU. No parallel computing was leveraged. For the four algorithms and two dimensions, Figure 4 plots the mean integration time as a function of maximum integral evaluations. Here, due to its precomputation, the NNI algorithm has a consistent integration time – regardless of the number of function evaluations. Conversely, the integration time of the other algorithms increases as the maximum evaluations increases.

These results are as expected, but highlight the time complexity of the algorithms. TRAPZ, MC, and ADAPT scale approximately according to $O(N)$, whereas the evaluation of NNI is completely decoupled from the original number of evaluation points. Instead, the time complexity of NNI is related to the number of nodes and dimensionality as $O(2^d k)$, which can be deduced from (18). The point where NNI becomes more efficient than the other methods depends on the relative complexity of a single integrand evaluation $C_f$ to a single polylogarithm evaluation $C_{Li}$, as

$$2^d k < N \frac{C_f}{C_{Li}}.$$  \hspace{1cm} (39)

To make TRAPZ, MC, or ADAPT more viable, it is, of course, possible to reduce the computational complexity $C_f$ through a similar pre-computation of the values of the function over $\mathcal{O}$, e.g., a lookup table. However, this approach
has the disadvantage of requiring large memory, which precludes embedded hardware applications. Additionally, if the goal is to integrate \( f(x) \) repeatedly over varying \( S \subseteq \mathcal{O} \), then it presents problems for algorithms that require integral evaluations at specific points, such as quadrature methods. Function evaluations at these points would then require \( n \)-dimensional interpolation methods, which will reduce the integration accuracy and increase the complexity further. So, if \( C_f \) is high, and the integration must nonetheless be evaluated quickly, or on basic hardware, NNI provides a more attractive alternative.

Finally, we note that for this testing, the polylogarithm function evaluations were evaluated to double floating-point accuracy (53-bit accuracy, or approximately 16 digits), and no parallel processing was leveraged. Additional or fewer digits can be computed as needed, with a corresponding computational cost of a polylogarithm evaluation. So, if less accuracy is expected (for example, if \( N \) was low), the accuracy of these evaluations could be reduced, which would decrease the evaluation time of NNI. Furthermore, the network structure of the NNI integration formulae are also in a convenient form for parallel processing. Leveraging this structure would allow for decreased processing time on multi-core systems, GPUs, FPGAs, or other specialized hardware.

IV. APPLICATIONS

For most numerical integration problems, NNI may not be the correct algorithm. The computational effort required to train a neural network, in most cases, outweighs the cost of standard algorithms. However, NNI is ideal in problems where its pre-computability can be leveraged, e.g., when an integrand function must be repeatedly integrated across different spaces, or when these integral evaluations must be done very quickly and with low memory usage. Thus, we propose NNI as a suitable integration algorithm for real-time or embedded applications, optimization, and other repetitive integration tasks – especially when the time complexity of the integrand function itself, \( C_f \), is high. In particular, the application of NNI to the optimization of integral functions is explored in the following section.

A. APPLICATIONS TO OPTIMIZATION OF INTEGRAL FUNCTIONS

The current section briefly illustrates some of the foundational aspects of how NNI can be applied to certain optimization problems. We show how the notation from Section II can be extended for optimization problems, and demonstrate how an NNI problem formulation can give access to the optimization gradient of the problem, allowing for the use of more efficient optimization algorithms. Finally, we illustrate the process with a simple example.

Consider the following constrained optimization problem, building on the notation of Section II,

\[
\max_{\theta \mid S(\theta) \subseteq \mathcal{O}} I(f, \theta) \tag{40}
\]

where the integration region \( S \) is now parameterized by an optimization vector \( \theta \), as

\[
S(\theta) = \left\{ x \in \mathcal{O} \mid A_1(\theta)x \leq b_1(\theta), A_2(\theta)x = b_2(\theta) \right\}, \tag{41}
\]

and the objective function \( I(f, \theta) \) is defined as the definite integral of \( f \) over \( S \), as

\[
I(f, \theta) = \int_{S(\theta)} f(x) \, dx. \tag{42}
\]

In other words, the goal is to find \( \theta \) which parameterizes a set \( S(\theta) \) which maximizes the definite integral of \( f \) over its volume. If the nature of \( f(x) \) requires the integration in \( (42) \) to be performed numerically and the dimensionality of \( S(\theta) \) is large, then the problem can become intractable using standard numerical integration techniques. However, this problem fits well into the scope of NNI, as it requires a large number of integral evaluations across various \( S(\theta) \subseteq \mathcal{O} \). By Theorem 1, we can find an approximated \( \hat{f}(x) \) such that

\[
\hat{I}(f, \theta) = \int_{S(\theta)} \hat{f}(x) \, dx. \tag{43}
\]

If \( \hat{f}(x) \) is trained to approximate \( f \) within \( \mathcal{O} \), then each optimization iteration can be performed tractably using NNI, even if the dimensionality of \( S(\theta) \) is large. Therefore, even in problems of multi-objective optimization where a Pareto front is being sought, the significant number of integration calls required does not cause the problem to become significantly less tractable. Additionally, standard numerical integration methods do not lend themselves well to optimization, as their discrete nature does not give one easy access to the optimization gradient. Without this gradient, optimization solvers are often less efficient and less robust [26]. With NNI, however, one has the added advantage that the optimization gradient is readily approximated from the network itself.

**Corollary 4:** For \( \hat{I}(f, \theta) \) as defined in \( (43) \), a closed-form of an approximate optimization gradient \( \nabla_\theta \hat{I}(f, \theta) \) exists, so long as the gradient of each of the elements of \( A_1, b_1, A_2, \) and \( b_2 \) can also be determined or estimated.

**Proof:** The closed-form of \( \hat{I}(f, \theta) \) after expanding the integrations through Theorem 2 and corollaries 2 and 3 is

\[
\hat{I}(f, \theta) = K_1k_1(\theta) \cdots k_d(\theta)
\]

\[
+ \sum_{j=1}^{K} [K_2\lambda_1(\theta) \cdots \lambda_d(\theta)] + K_3 \lambda_i \left( -e^{\eta(\theta)} \right), \tag{44}
\]

where \( K_1 \ldots K_3 \) are constants, and \( \eta(\theta) \) are functions that consist of a linear combination of the elements \( A_{ij}, b_{ij}, A_{2ij} \) and \( b_{2ij} \) of the constraint matrices \( A_1, b_1, A_2, \) and \( b_2 \). If the gradient w.r.t \( \theta \) of these matrix elements, \( \nabla_{\theta} A_{ij}, \nabla_{\theta} b_{ij}, \nabla_{\theta} A_{2ij} \) and \( \nabla_{\theta} b_{2ij} \) can be analytically determined or easily estimated, then the gradients \( \nabla_{\theta} \lambda_i(\theta), \nabla_{\theta} \lambda_j(\theta), \nabla_{\theta} \eta(\theta) \) can be determined by the linearity property of the gradient operator. Furthermore, the gradient of the polylogarithm term can be determined through chain rule as

\[
\nabla_\theta \lambda_i \left( -e^{\eta(\theta)} \right) = (\nabla_\theta \eta(\theta)) \lambda_i - \left( -e^{\eta(\theta)} \right). \tag{45}
\]
Thus, the entire expression is comprised of the sum of products of functions whose gradients exist in closed-form. Then, by the product rule, a closed-form of the entire gradient expression exists.

Since the gradient of the objective function can be estimated, gradient-based methods can also be applied to the problem. Additionally, the logic of Theorem 4 applies equally well to the gradient $\nabla_I \hat{I}(f, \theta)$ as it does to the integral $\hat{I}(f, \theta)$ itself. Therefore, the optimization Hessian can also be estimated, if desired, which gives one easy access to second-order optimization methods.

Example 5 (Positioning of an n-Dimensional Hyperrectangle): Recall the integration of an n-dimensional hyperrectangle in Example 3. Suppose we have the optimization problem of selecting the coordinates of the center $\theta \in \mathbb{R}^n$ of a hyperrectangle with fixed side lengths $d \in \mathbb{R}^n$, such that $I(f, \theta)$ is maximized. Mathematically, the problem could be formulated as

$$\max_{\theta} I(f, \theta), \quad I(f, \theta) = \int_{S(\theta)} f(x) dx, \quad (46)$$

with

$$S(\theta) = \left\{ x \in \mathcal{O} \mid \begin{bmatrix} -1 \\ I \\ 1 \end{bmatrix} x \preceq \begin{bmatrix} -\theta + d/2 \\ \theta + d/2 \end{bmatrix} \right\}, \quad (47)$$

where $I$ is the $n \times n$ identity matrix. The integrated form of $\hat{I}(f, \theta)$ is the same as in Example 3. Though multiple applications of the chain and product rules, the optimization gradient of $\hat{I}(f, \theta)$ is

$$\nabla_I \hat{I}(f, \theta) = b^{(2)} C + \sum_{j=1}^{k} w_j^{(2)} \left[ C + \frac{\partial \Phi_j}{\partial \theta} \right], \quad (48a)$$

where the $n$-vectors $C$ and $\nabla_I \Phi_j$ are defined by

$$C = \sum_{i=1}^{n} \left[ (\nabla_I \beta_i - \nabla_I \alpha_i) \prod_{g=1}^{n} (\nabla_I \beta_g - \nabla_I \alpha_g) \right], \quad (48b)$$

$$\nabla_I \Phi_j = \sum_{r=1}^{2^n} \left\{ -\xi_r \left( \sum_{i=1}^{n} w_i^{(1)} \nabla_I \ell_{i,r} \right) \right\}. \quad (48c)$$

The findings show NNI to be a viable method for the rapid computation of numerical integrals in situations where the definite integral of a function across different bounds must be evaluated multiple times. It is proposed for real-time or embedded applications, particularly when the time complexity of the integrand function itself, $C_r$, is high. Furthermore, we explored the application of NNI to the optimization of integral functions, in which it was shown that NNI has the benefit of allowing for easy estimation of the optimization gradient. Thus, NNI is presented as an algorithm allowing for fast or real-time evaluations or even optimization of difficult, high dimension integrations.

V. CONCLUSION

We present a novel numerical integration technique, Neural Network Integration, or NNI. A shallow neural network design was used to approximate the integrand function within a bounded set. This network was designed in such a way to allow for a closed-form solution to the definite integral across any generalized polyhedron in the domain of $\hat{f}$. Thus, after training the neural network, the definite integral across varying spaces can be computed analytically – essentially allowing for a “pre-computation” of the integration problem.

The proposed method was tested for accuracy and evaluation time using the Genz [27] function families and compared against uniform sampling numerical integration algorithms TRAPZ (trapezoidal integration) and MC (Monte Carlo integration), as well as the adaptive quadrature routine ADAPT [5]. NNI was shown to be a viable method of integration, giving comparable accuracy as the other tested uniform-sampling methods. Its accuracy lagged compared to the adaptive routine ADAPT on functions with high variability over the integration domain. However, it gave good results on functions with constant variability. These results suggest that further research into the application of adaptive sampling of the integrand function, or intelligent weighting of the sample points during network training, could give further improvements to the accuracy of NNI integral estimates. Finally, NNI was also found to perform particularly well compared to the other algorithms in high dimensions, when the number of integrand evaluations was limited.

The integration time of each method was compared and confirmed that NNI results can be computed quickly, regardless of the number of sampling points. A comparison of the time complexity of NNI and standard numerical integration algorithms gave a guideline for determining if NNI or standard algorithms are best-suited to an application.

The findings show NNI to be a viable method for the rapid computation of numerical integrals in situations where the definite integral of a function across different bounds must be evaluated multiple times. It is proposed for real-time or embedded applications, particularly when the time complexity of the integrand function itself, $C_r$, is high. Furthermore, we explored the application of NNI to the optimization of integral functions, in which it was shown that NNI has the benefit of allowing for easy estimation of the optimization gradient. Thus, NNI is presented as an algorithm allowing for fast or real-time evaluations or even optimization of difficult, high dimension integrations.

REFERENCES

[1] H. Niederreiter, Random Number Generation and Quasi-Monte Carlo Methods. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, Jan. 1992.
[2] I. H. Sloan, “Lattice methods for multiple integration,” J. Comput. Appl. Math., vols. 12–13, pp. 131–143, May 1985.
[3] T. Gerstner and M. Griebel, “Numerical integration using sparse grids,” Numer. Algorithms, vol. 18, pp. 209–232, Mar. 1998.
[4] E. Novak and K. Ritter, “High dimensional integration of smooth functions over cubes,” Numer. Math., vol. 75, no. 1, pp. 79–97, Nov. 1996.
[5] A. C. Genz and A. A. Malik, “Remarks on algorithm 006: An adaptive algorithm for numerical integration over an N-dimensional rectangular region,” J. Comput. Appl. Math., vol. 6, no. 4, pp. 295–302, Dec. 1980.
[6] P. van Dooren and L. de Ridder, “An adaptive algorithm for numerical integration over an N-dimensional cube,” J. Comput. Appl. Math., vol. 2, no. 3, pp. 207–217, Sep. 1976.
[7] G. P. Lepage, “A new algorithm for adaptive multidimensional integration,” J. Comput. Phys., vol. 77, no. 2, pp. 192–203, May 1978.
[8] W. H. Press and G. R. Farrar, “Recursive stratified sampling for multidimensional Monte Carlo integration,” Comput. Phys., vol. 4, no. 2, p. 190, 1990.

[9] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1998.

[10] S. Ferrari and R. F. Stengel, “Smooth function approximation using neural networks,” IEEE Trans. Neural Netw., vol. 16, no. 1, pp. 24–38, Jan. 2005.

[11] Z. Zhe-Zhao, W. Yao-Nan, and W. Hui, “Numerical integration based on a neural network algorithm,” Comput. Sci. Eng., vol. 8, no. 4, pp. 42–48, Jul. 2006.

[12] H. Wu, “Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions,” Inf. Sci., vol. 179, no. 19, pp. 3432–3441, Sep. 2009, doi: 10.1016/j.ins.2009.06.006.

[13] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Math. Control. Signals, Syst., vol. 2, no. 4, pp. 303–314, Dec. 1989.

[14] L. Vinet and A. Zhedanov, “A ‘missing’ family of classical orthogonal polynomials,” Neural Netw., vol. 2, pp. 359–366, Nov. 2010.

[15] R. Crandall. (2006). Note on Fast Polylogarithm Computation. [Online]. Available: http://www.reed.edu/~crandall/papers/Polylog.pdf

[16] F. Johansson. (2013). mpmath: A Python Library for Arbitrary-Precision Floating-Point Arithmetic (Version 0.18). [Online]. Available: http://mpmath.org

[17] M. Hazewinkel, “Surface integral,” in Encyclopedia of Mathematics. Dordrecht, The Netherlands: Springer, 2001.

[18] J. F. Mas and J. J. Flores, “The application of artificial neural networks to the analysis of remotely sensed data,” Int. J. Remote Sens., vol. 29, no. 3, pp. 617–663, Feb. 2008.

[19] G.-B. Huang, “Learning capability and storage capacity of two-hidden-layer feedforward networks,” IEEE Trans. Neural Netw., vol. 14, no. 2, pp. 274–281, Mar. 2003.

[20] E. B. Baum and D. Haussler, “What size net gives valid generalization?” Neural Comput., vol. 1, no. 1, pp. 151–160, 2008.

[21] M. F. Moller, “A scaled conjugate gradient algorithm for fast supervised learning,” Neural Netw., vol. 6, no. 4, pp. 525–533, Jan. 1993.

[22] D. J. C. MacKay, “A practical Bayesian framework for backpropagation networks,” Neural Comput., vol. 4, no. 3, pp. 448–472, May 1992.

[23] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” J. Soc. Ind. Appl. Math., vol. 11, no. 2, pp. 431–441, Jun. 1963.

[24] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: The RPROP algorithm,” in Proc. IEEE Int. Conf. Neural Netw., Mar./Apr. 1993, pp. 586–591.

[25] O. Ki and E. Uncuo, “Comparison of three back-propagation training algorithms for two case studies,” Indian J. Eng. Mater. Sci., vol. 12, pp. 434–442, Oct. 2005.

[26] MathWorks, Natick, MA, USA. (2018). Optimization Toolbox User’s Guide. [Online]. Available: https://www.mathworks.com/help/pdf_doc/optim/optim_tb.pdf

[27] A. Genz, “Testing multidimensional integration routines,” in Tools, Methods and Languages for Scientific and Engineering Computation, New York, NY, USA: Elsevier, 1984, pp. 81–94.

[28] A. Genz, “A package for testing multiple integration subroutines,” in Numerical Integration. Dordrecht, The Netherlands: Springer, 1987, pp. 337–340.