scientific reports

Check for updates

OPEN

# A machine learning approach for efficient multi-dimensional integration

Boram Yoon

**Many physics problems involve integration in multi-dimensional space whose analytic solution is not available. The integrals can be evaluated using numerical integration methods, but it requires a large computational cost in some cases, so an efficient algorithm plays an important role in solving the physics problems. We propose a novel numerical multi-dimensional integration algorithm using machine learning (ML). After training a ML regression model to mimic a target integrand, the regression model is used to evaluate an approximation of the integral. Then, the difference between the approximation and the true answer is calculated to correct the bias in the approximation of the integral induced by ML prediction errors. Because of the bias correction, the final estimate of the integral is unbiased and has a statistically correct error estimation. Three ML models of multi-layer perceptron, gradient boosting decision tree, and Gaussian process regression algorithms are investigated. The performance of the proposed algorithm is demonstrated on six different families of integrands that typically appear in physics problems at various dimensions and integrand difficulties. The results show that, for the same total number of integrand evaluations, the new algorithm provides integral estimates with more than an order of magnitude smaller uncertainties than those of the VEGAS algorithm in most of the test cases.**

Monte Carlo integration is a numerical method evaluating the integral of an integrand over a finite region using random sampling. As a consequence of the random sampling, in contrast to deterministic methods, the result of the Monte Carlo integration is an estimate of the true value that comes with a statistical uncertainty. For higher-dimensional integral problems, the Monte Carlo integration methods provide smaller uncertainties than deterministic methods, such as the trapezoidal rule[1], for a given number of integrand evaluations. As a result, the Monte Carlo integration methods are broadly used in numerous physics calculations that involve numerical integrations[2–6].

The most widely used strategies reducing the variance of the Monte Carlo integration estimate are *importance sampling* and *stratified sampling*. These strategies are implemented in many algorithms, such as the VEGAS[7–9] and MISER[1]. Recently, an idea utilizing generative machine learning (ML) models to perform the importance sampling is also proposed[10].

In this paper, we present a novel algorithm numerically evaluating multi-dimensional integrals. The new algorithm involves computations for the training of and prediction with the ML algorithms, in addition to the evaluation of the integrand. Assuming that evaluations of the integrand is computationally much more expensive than the ML calculations, throughout the paper, we focus on reducing the variance of an integral estimate for a given number of integrand evaluations.

## Method

Suppose that we have a ML regression model $\tilde{f}(\mathbf{x})$ that approximates a multi-dimensional function $f(\mathbf{x}) \approx \tilde{f}(\mathbf{x}) \equiv \tilde{y}$. Here $\mathbf{x}$ is the input vector of the regression model, which is called the *independent variable*, and $\tilde{y}$ is the output of the regression model, which is called the *dependent variable*. The integral of $f(\mathbf{x})$ over an integration region $\Omega \in \mathbb{R}^D$ can be split into two integrals.

$$I = \int_\Omega f(\mathbf{x}) d\mathbf{x} \tag{1}$$

CCS-7, Computer, Computational and Statistical Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA. email: boram@lanl.gov

$$= \underbrace{\int_\Omega \tilde{f}(\mathbf{x})d\mathbf{x}}_{I_{(a)}} + \underbrace{\int_\Omega \left(f(\mathbf{x}) - \tilde{f}(\mathbf{x})\right)d\mathbf{x}}_{I_{(b)}} . \tag{2}$$

Here the $I_{(a)}$ is an integration of the ML regression model, which does not require an evaluation of $f(\mathbf{x})$ to calculate. Depending on the regression model, the integral $I_{(a)}$ may be calculated analytically or numerically. Since $\tilde{f}(\mathbf{x})$ approximates $f(\mathbf{x})$, the $I_{(a)}$ provides an approximation of the target integral $I$. The $I_{(b)}$ in Eq. (2) provides a correction to the bias of the approximation in $I_{(a)}$. The integral can be evaluated using a numerical method, such as the Monte Carlo integration. In the Monte Carlo integration, variance of the $I_{(b)}$ is proportional to

$$\mathrm{Var}\left[f(\mathbf{X}) - \tilde{f}(\mathbf{X})\right] = \mathrm{Var}\left[f(\mathbf{X})\right] + \mathrm{Var}\left[\tilde{f}(\mathbf{X})\right] - 2\mathrm{Cov}\left[f(\mathbf{X}), \tilde{f}(\mathbf{X})\right] \tag{3}$$

$$\approx 2\mathrm{Var}\left[f(\mathbf{X})\right]\left(1 - \mathrm{Corr}\left[f(\mathbf{X}), \tilde{f}(\mathbf{X})\right]\right), \tag{4}$$

where the second line assumes a good ML regression model that gives $\mathrm{Var}\left[\tilde{f}(\mathbf{X})\right] \approx \mathrm{Var}\left[f(\mathbf{X})\right]$. Therefore, the $I_{(b)}$ can be estimated precisely with a small number of Monte Carlo samples when correlation between $f(\mathbf{X})$ and $\tilde{f}(\mathbf{X})$ is high. The total uncertainty of the integral, $\sigma_I$, can be calculated by combining the errors of the two terms: $\sigma_I^2 = \sigma_{I_{(a)}}^2 + \sigma_{I_{(b)}}^2$. The idea replacing an observable by its approximation with a proper correction term was used in the field of lattice quantum chromodynamics[11–13]. In this paper, we generalize the idea for multi-dimensional integral problems using ML regression algorithms.

Note that the equalities in Eqs. (1) and (2) hold independently of the regression accuracy. When $\tilde{f}(\mathbf{x})$ poorly approximates $f(\mathbf{x})$, the $I_{(b)}$ becomes difficult to calculate, and a numerical evaluation of $I_{(b)}$ yields large $\sigma_{I_{(b)}}^2$ for a given number of integrand evaluations. Therefore, this approach always provides a correct error estimation with an unbiased expectation value of the integral, provided good integration algorithms for the evaluation of $I_{(a)}$ and $I_{(b)}$. However, a good ML regression model accurately describing the integrand is essential for obtaining the integral estimate with a small statistical uncertainty.

**Machine learning models.** In this paper, we examine three regression algorithms: Multi-layer Perceptron (MLP), Gradient Boosting Decision Tree (GBDT), and Gaussian Process (GP). MLP is a feedforward artificial neural network that produces outputs from inputs based on multiple layers of perceptrons[14]. The model is flexibly applicable to various kinds of data and scales well up to a large number of data. GBDT is a sequence of shallow decision trees such that each successive decision tree compensates for the prediction error of its predecessor[15,16]. The model provides a good regression performance with no complicated tuning of hyperparameters and pre-processing of training data. An integration of the GBDT regression models can be calculated analytically because the model is simply a set of intervals of input variables and their output values[10]. GP regression is a nonparametric model that finds an optimal covariance kernel function explaining training data[17]. The model is good at interpolating the observations and works well with a small dataset. Analytic integrability of the regression model depends on kernel choice. For example, in the case of the Radial Basis Function (RBF) kernel, which is one of the most popular kernels in GP, prediction of an input vector is given by the dot product of a Gaussian function of the input vector and a constant vector, as described in Eq. (7), so its analytic integration is given by error functions.

**Training data.** Building a ML regression model approximating $f(\mathbf{x})$ requires training samples of $\{(\mathbf{x}_i, f(\mathbf{x}_i))\}_{i=1}^{N_{\mathrm{train}}}$. To minimize the prediction error for a given number of training data, $N_{\mathrm{train}}$, it is essential to collect the training samples near the peaks of the function (importance sampling) and where the function changes rapidly (stratified sampling). Such training data can be sampled by utilizing conventional numerical integration algorithms, such as the VEGAS, which includes efficient sampling algorithms based on the importance sampling and stratified sampling. When the peaks of the function are localized, the training samples obtained using VEGAS build a much more accurate ML regression model than those from a uniform sampling method.

**Data scaling.** Many ML regression algorithms benefit from scaling the dependent variable. Especially when the dependent variable varies by orders of magnitude within the range of interest, which is a typical situation in difficult multi-dimensional integral problems, the data scaling plays a crucial role in obtaining a good regression performance. The most widely used scaling algorithms are min-max scaling and standardization:

$$y' = \frac{y - \min(y)}{\max(y) - \min(y)} \quad \text{[Min-max scaling]}, \qquad y' = \frac{y - \overline{y}}{\sigma_y} \quad \text{[Standardization]}, \tag{5}$$

where $y'$ is the scaled variable, $\min(y)$ and $\max(y)$ are the minimum and maximum of $y$, $\overline{y}$ is the average of $y$, and $\sigma_y$ is the standard deviation of $y$. For the data with large scale variation, however, these scaling methods are dominated by the data of large magnitude and lose sensitivity to the data of small values.

To avoid the scale issue, we use the nth-root scaling defined as

$$y' = \mathrm{sgn}(y) \cdot |y|^{1/n}, \tag{6}$$

where $\mathrm{sgn}(y)$ is the sign of $y$, and $n$ is a positive integer. This is a strictly monotonic transformation whose inverse is $y = \mathrm{sgn}(y') \cdot |y'|^n$. When $n$ is too small, the transformation will not remove the large scale variation that makes

ML algorithms difficult to fit, and when $n$ is too large, the transformation will wash out (or flatten) the information of the data that is essential for training a ML algorithm. Therefore, an optimal choice of $n$ is important in obtaining a good regression model and is different for different ML algorithms. The optimal value of $n$ can be obtained using the training data. Taking a small portion (e.g. 10–50%) of the training data as a validation dataset, one can train a regression model on the remaining training samples with various choices of $n$ and find the optimal value of $n$ that gives the minimum prediction error on the validation dataset. Once the $n$ is determined, a final regression model can be obtained using the full training data.

This nth-root scaling plays a crucial role in building a good regression model for most of the integral problems. In this study, we standardize the $y$ after the nth-root scaling to maximize the regression performance.

**Evaluation of $I_{(a)}$ and $I_{(b)}$.** When $n = 1$, $I_{(a)}$ of Eq. (2) can be calculated analytically for certain regression algorithms. When $n > 1$, however, the ML predictions should be processed by the inverse of the nth-root transformation, so the analytic integral becomes complicated. For example, GP regression with a RBF kernel for a nth-root scaled data can be written as

$$\tilde{f}'(\mathbf{x}) = \sum_{i=1}^{N_{\text{train}}} \alpha_i \exp\left[-\frac{1}{2l^2}||\mathbf{x} - \mathbf{x}_i||^2\right], \tag{7}$$

where $||\mathbf{x}||$ is the Euclidean norm of $\mathbf{x}$, $\mathbf{x}_i$ are the training data, and $\alpha_i$ and $l$ are constants that are determined from the training. To obtain the prediction of the integrand $\tilde{f}(\mathbf{x})$, the GP regression needs to be transformed as $\tilde{f}(\mathbf{x}) = \text{sgn}(\tilde{f}'(\mathbf{x}))|\tilde{f}'(\mathbf{x})|^n$. For a positive integrand, the power of $n$ of Eq. (7) can be expanded analytically, but the number of terms is large for large $N$ and $n$.

For simplicity, we use a numerical method, the VEGAS algorithm, to evaluate $I_{(a)}$ and $I_{(b)}$. Since the peaks of the $f(\mathbf{x})$ are flattened by subtracting the $\tilde{f}(\mathbf{x})$ in the integrand of $I_{(b)}$, a simple Monte Carlo integration works well for $I_{(b)}$. However, the VEGAS outperforms the simple Monte Carlo integration when the regression is not accurate enough.

## Numerical experiments

In this section, we present numerical experiments of the proposed integration algorithm using ML. The precisions of the integral estimates are compared with those of the VEGAS algorithm, which is one of the best performing algorithms on the market[18], at a similar number of integrand evaluations.

**Test integrands.** In order to test the performance of the numerical integration, we use the six families of the integrands proposed in Ref.[19] that typically appear in physics problems:

$$\begin{aligned}
f_1(\mathbf{x}) &= \cos(2\pi w_1 + \mathbf{c}\cdot\mathbf{x}) && \text{[Oscillatory]}\,,\\
f_2(\mathbf{x}) &= \prod_{i=1}^{D} \frac{1}{c_i^{-2} + (x_i - w_i)^2} && \text{[Product peak]}\,,\\
f_3(\mathbf{x}) &= \frac{1}{(1 + \mathbf{c}\cdot\mathbf{x})^{D+1}} && \text{[Corner peak]}\,,\\
f_4(\mathbf{x}) &= \exp(-\textstyle\sum_{i=1}^{D} c_i^2 (x_i - w_i)^2) && \text{[Gaussian]}\,,\\
f_5(\mathbf{x}) &= \exp(-\mathbf{c}\cdot|\mathbf{x} - \mathbf{w}|) && [C^0 - \text{function}]\,,\\
f_6(\mathbf{x}) &= \begin{cases} 0 & \text{if } x_1 > w_1 \text{ or } x_2 > w_2\,,\\ \exp(\mathbf{c}\cdot\mathbf{x}) & \text{otherwise.}\end{cases} && \text{[Discontinuous]}\,.
\end{aligned} \tag{8}$$

Here, $D$ is the dimension of $\mathbf{x}$, and $w_i \in [0, 1)$ is the parameter that is supposed to shift the peaks of the integrand without changing the difficulty of the integral problem. One exception is $f_6(\mathbf{x})$, as the small value of $w_1$ or $w_2$ makes the function to be localized in small region and makes the integral problem difficult. To avoid the unwanted effect, we restrict $w_i \in [0.1, 0.9)$ for $f_6(\mathbf{x})$. $c_i$ is a positive parameter that controls the difficulty of the integral. In general, increasing the value of $c_i$ increases the difficulty of the integral problem. To fix the difficulty of the integral, we randomly choose $c_i$ from a uniform distribution in [0, 1) and renormalize the vector by multiplying a constant factor so that $||\mathbf{c}||_1 = \sum_i |c_i|$ becomes the target constant. In this study, we carry out the integration for 36 different random choices of $\mathbf{w}$ and $\mathbf{c}$ and take average performance. To fix the integration difficulty, we normalize $\mathbf{c}$ to three different values of $||\mathbf{c}||_1 = 1, 3$, and 8. Integration is performed in a $D$-dimensional unit hypercube, and the results are compared at three different dimensions of $D = 5, 8$, and 10.

**ML regression algorithms and hyperparameters.** For the implementation of the MLP, GP, and GBDT regression algorithms, we use the scikit-learn python library[20]. For MLP, four hidden layers of 128, 128, 128, and 16 neurons with rectified linear unit (ReLU) activation functions are used. Training is performed using Adam optimization algorithm[21] with the learning rate of $10^{-4}$. Training updates are performed until there is no decrease of the validation score with a tolerance of $10^{-6}$ for 20 epochs with 10% of validation fraction. For GP, RBF with a constant kernel is used, and length scale and constant are determined using L-BFGS-B optimizer[22]. For GBDT, we use 1000 weak estimators with a learning rate of 0.01 and a subsampling ratio of 0.3. The maximum depth of each decision tree is limited to 4. Note that here we use a relatively large number of estimators with a small subsampling ratio so that the regression output becomes a smooth function in $\mathbf{x}$. In this proof-of-principal study, we did not explore the extensive phase space of the hyperparameters but took the best solution among the few choices around the default values of the scikit-learn library[20] we tried.

The powers of the nth-root scaling $n \in [1, 50]$ for MLP and GP regressions are determined by using 20% of training data as a validation dataset. The performance of the GBDT algorithm is not very sensitive to the scaling but $n > 1$ gives better performance than the $n = 1$ case. So, we use a fixed number $n = 3$ for GBDT regression.

**VEGAS setup.** For the VEGAS numerical integration, we use Lepage's VEGAS python library version 3.4.5[9,23]. The library has two damping parameters: $\alpha$ and $\beta$. The parameter $\alpha$ controls the remapping of the integration variables in the VEGAS adaptation. A smaller value gives the slower grid adaption for a conservative estimate. Here, we use $\alpha = 0.5$, which is the default value of the library, for most of the calculations. One exception is the discontinuous integrand family $f_6(\mathbf{x})$, which is more difficult to evaluate than other integrand families and requires a large number of samples per iteration or slow grid adaptation to converge to the exact integral solution. To make the VEGAS integral stable, we use $\alpha = 0.2$ for $f_6(\mathbf{x})$. The parameter $\beta$ controls the redistribution of integrand evaluations in the stratified sampling. $\beta = 1$ is the theoretically optimal value, and $\beta = 0$ means no redistribution. Here, we use $\beta = 0.75$, which is the default value of the library.

Another important parameters are the number of iterations for the VEGAS grid adaptation ($N_{\text{itn}}$) and the approximated number of integrand evaluations per iteration ($N_{\text{eval}}$). These parameters are set differently for different VEGAS tasks performed in this study: (1) calculation of the target integral $I$ in Eq. (1) for a comparison, (2) sampling the training data, (3) calculation of $I_{(\text{b})}$ in Eq. (2), and (4) calculation of $I_{(\text{a})}$ in Eq. (2).

- In task (1), we use $N_{\text{itn}} = 20$ at two different values of $N_{\text{eval}} = 500$, and 1000. When $\beta = 0$, the total number of integrand evaluations will be $N_{\text{itn}} \times N_{\text{eval}}$. Because of the redistribution that happens when $\beta > 0$, however, the total number of integrand evaluations for this task drops to around $N \approx N_{\text{itn}} \times N_{\text{eval}}/2$ for the test functions used in this study.
- In task (2), we use $N_{\text{itn}} = 10$ for the most of the integrand families with the same $N_{\text{eval}}$ values used in task (1). In this task, all the integrand calls, $\big(\mathbf{x}, f(\mathbf{x})\big)$, are collected as the ML training data. The total number of integrand evaluations in this task is $N_{\text{train}} \approx N/2$. Two exceptions are $f_3(\mathbf{x})$, where we use $N_{\text{itn}} = 14$, and $f_6(\mathbf{x})$, where we use $N_{\text{itn}} = 6$, which are the choices that stabilize the VEGAS integration of task (3). The choice of $N_{\text{itn}}$ determines the ratio of the number of integrand evaluations for the training ($N_{\text{train}}$) and bias correction ($N_{\text{crxn}}$). The parameter can be tuned for a given problem so that it minimizes the integral uncertainty. In this proof-of-principal study, however, we take $N_{\text{itn}} = 10$, which makes the ratio $N_{\text{train}} : N_{\text{crxn}} \approx 1 : 1$, as our default value and change it only when we find an instability in the VEGAS integration of task (3).
- In task (3), our target total number of integrand evaluations is $N - N_{\text{train}}$, so that the total number of integrand evaluations in the ML integrator is the same as that of the VEGAS integration in task (1). To make $N_{\text{crxn}}$ as close as possible to $N - N_{\text{train}}$, we set $N_{\text{eval}} = (N - N_{\text{train}})/5$ with $\alpha = 0.5$ and carry out the VEGAS iterations until the accumulated number of integrand evaluations becomes greater than or equal to $N - N_{\text{train}}$, saving the results for each iteration, separately. Then, we find the iteration number that gives the accumulated number of integrand evaluations closest to $N - N_{\text{train}}$ and take the integral estimate at the number of iterations as our final results.
- In task (4), we use $N_{\text{itn}} = 30$ and set $N_{\text{eval}}$ to those used in task (1) multiplied by a factor of 1000. We stop the VEGAS iteration when the error of $I_{(\text{a})}$ becomes smaller than 20% of the error of $I_{(\text{b})}$. In this study, we use a numerical integration method to evaluate $I_{(\text{a})}$ so that all results for different integrand families and ML algorithms could be obtained from the same condition. However, a numerical approach requires evaluation of the ML model $\tilde{f}(\mathbf{x})$, so it provides a cost reduction only when the evaluation of the ML model is cheaper than the evaluation of the target integrand $f(\mathbf{x})$. We recommend using an analytic approach for the evaluation of $I_{(\text{a})}$ whenever it is available. When analytic approach is not available or computationally expensive due to a large scaling power, $N_{\text{eval}}$ should be tuned considering the evaluation cost of $\tilde{f}(\mathbf{x})$ and the integral precision required.

VEGAS integral estimates are obtained by taking a weighted average of the estimates from each VEGAS iteration. Whenever the p-value of the weighted average is smaller than 0.05, we discard the results and rerun the VEGAS integration with a different random seed.

**Results.** Table 1 shows the precision gain of the proposed integration algorithm over VEGAS.

$$\text{Gain} = \frac{\sigma_I \text{ of VEGAS}}{\sigma_I \text{ of ML Integrator}}. \tag{9}$$

Total number of integrand evaluations of the ML integrator ($N_{\text{train}} + N_{\text{crxn}}$) is similar to that of the VEGAS integration ($N$). The full list of $N$, $N_{\text{train}}$, $N_{\text{crxn}}$, and precision of the integral algorithms are given in Supplementary Tables S1 to S6.

The best performing ML algorithms are GP for the integrand families 1–4, MLP for the integrand family 5, and GBDT for the integrand family 6. Figure 1 clearly explains these results: GP with a RBF kernel shows very good performance in describing smooth functions but fails in $C^0$ and discontinuous functions. MLP shows mediocre performance for the all functional forms, and GBDT, which is a combination of the discrete decision trees, outperforms the MLP in describing the discontinuous integrands.

For all test cases, the ML integrator performs better than VEGAS. The gain is higher when $D$ is smaller and when $||\mathbf{c}||_1$ is smaller. Also, the gain tends to be increased when $N$ is larger, which indicates a better scaling behavior than VEGAS. In case of the integrations with the GP regression algorithm, the $\sigma_I$ of the ML integrator is up to four orders of magnitude smaller than that of VEGAS. When GP is efficiently applied, the difference

| Integrand family | | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| (ML Algorithm) | | | (GP) | (GP) | (GP) | (GP) | (MLP) | (GBDT) |
| N | D | $\|\|\mathbf{c}\|\|_1$ | | | | | | |
| ≈5000 | 5 | 1.0 | 407 (29) | 6181 (319) | 329 (4) | 6223 (258) | 14.1 (6) | 10.0 (5) |
| | | 3.0 | 197 (2) | 1321 (67) | 211 (98) | 1151 (47) | 5.5 (2) | 6.5 (4) |
| | | 8.0 | 181 (1) | 47 (8) | 138 (5) | 346 (9) | 3.1 (1) | 3.1 (2) |
| | 8 | 1.0 | 352 (28) | 8292 (262) | 207 (3) | 8918 (177) | 8.9 (3) | 8.6 (6) |
| | | 3.0 | 141 (2) | 1219 (100) | 130 (1) | 1606 (46) | 3.6 (1) | 4.3 (2) |
| | | 8.0 | 107 (3) | 20 (2) | 57 (3) | 319 (7) | 1.6 (1) | 2.3 (1) |
| | 10 | 1.0 | 345 (29) | 8533 (250) | 178 (2) | 8941 (189) | 6.5 (2) | 8.2 (4) |
| | | 3.0 | 119 (1) | 972 (106) | 99 (1) | 2028 (56) | 2.5 (1) | 4.3 (2) |
| | | 8.0 | 54 (2) | 17 (2) | 35 (1) | 386 (10) | 1.2 (1) | 2.0 (1) |
| ≈10000 | 5 | 1.0 | 357 (24) | 5770 (311) | 299 (3) | 5497 (235) | 16.7 (5) | 14.5 (7) |
| | | 3.0 | 175 (2) | 1241 (44) | 200 (92) | 1084 (44) | 7.9 (3) | 7.5 (3) |
| | | 8.0 | 157 (5) | 121 (20) | 144 (6) | 279 (8) | 4.0 (1) | 3.6 (2) |
| | 8 | 1.0 | 359 (28) | 9130 (307) | 215 (3) | 9649 (203) | 16.7 (5) | 11.0 (6) |
| | | 3.0 | 147 (2) | 1592 (89) | 142 (1) | 1698 (49) | 6.6 (2) | 5.0 (2) |
| | | 8.0 | 132 (2) | 40 (5) | 97 (3) | 320 (8) | 3.1 (1) | 2.7 (1) |
| | 10 | 1.0 | 339 (27) | 9092 (218) | 181 (2) | 9535 (118) | 13.0 (5) | 9.8 (6) |
| | | 3.0 | 118 (1) | 1679 (111) | 116 (1) | 1976 (47) | 5.1 (2) | 4.7 (3) |
| | | 8.0 | 89 (3) | 36 (4) | 57 (2) | 372 (9) | 2.4 (1) | 2.2 (1) |

**Table 1.** Precision gain of the proposed algorithm over VEGAS, defined in Eq. (9), at two different number of integrand evaluations ($N$) for three dimensions ($D$) and three integrand difficulties ($||\mathbf{c}||_1$) on the six integrand families listed in Eq. (8). The results are averaged over 36 random samples. The numbers in the parentheses are the standard deviation of the mean.

between the target integrand and its ML prediction is tiny, which makes the value and error of $I_{(b)}$ small. As a result, the final error is dominated by the error of $I_{(a)}$, which can be improved without increasing the number of $f(\mathbf{x})$ evaluations. As an example, $I_{(a)}$ and $I_{(b)}$ of the integrands shown in Fig. 1 are given below:

$$\begin{array}{lll} I & = I_{(a)} & + I_{(b)} \\ f_4: 0.3195642(12) & = 0.3195644(12) & - 0.000000189(30) \quad [GP], \\ f_5: 0.11582(10) & = 0.115408(17) & + 0.00041(10) \quad [MLP], \\ f_6: 14.463(42) & = 13.8934(51) & + 0.570(42) \quad [GBDT]. \end{array} \tag{10}$$

Since the ML integrator uses VEGAS, it inherits the potential instability of the VEGAS for small $N_{\text{eval}}$, which introduces a systematic bias in the integration results. In general, the instability can be avoided by increasing $N$ or decreasing the value of $\alpha$; a detailed description of how to deal with the instability is given in Ref.[24]. For the ML integrator, the most fragile part is the integration for $I_{(b)} = \int (f - \tilde{f}) d\mathbf{x}$. When such instability is observed, for a given $N$, one can increase the ratio of $N_{\text{train}}$ for a better prediction or increase the ratio of $N_{\text{crxn}}$ for a more stable integrand evaluation, depending on the integral problem. It is also important to use a ML regression algorithm that yields a smooth $\tilde{f}(\mathbf{x})$. As shown in the right column of Fig. 1, a non-smooth $\tilde{f}(\mathbf{x})$, such as the one from GBDT, makes $f(\mathbf{x}) - \tilde{f}(\mathbf{x})$ highly oscillating and the integral difficult to evaluate. Among the three regression algorithms used in this study, we find that the smooth prediction of GP gives the most stable integration for $I_{(b)}$. To check the instability, we do not manually tune the integration parameters for each integral problem but use a general setting for most of the calculations. As a result, we could observe a few integral results deviating from the true answer by more than $4\sigma$, mostly in case the integral families 3 and 6. It shows that the ML integrator is more stable than VEGAS for the integral family 6, but less stable for the integral family 3 mainly due to the non-smooth prediction of the MLP and GBDT. Since the number of more than $4\sigma$ deviations is small compared to the total number of random samples, inclusion of those occurrences does not change the average results, so we did not exclude those occurrences from our average results. The number of more than $4\sigma$ deviations for each integral problem is given in Supplementary Tables S1 to S6.

Table 2 gives a comparison of the training and prediction cost of the wallclock time for different ML models. For the prediction cost, we measure the regression wallclock time of trained models on $10^5$ random samples. It shows that training a ML algorithm is much more expensive than making predictions using a trained ML model. It also shows that GP is more expensive than MLP and GBDT. Note that the ML training and prediction costs can be reduced significantly by changing the model parameters at the cost of slightly worse prediction ability. For example, MLP cost can be reduced by increasing the learning rate and lowering the number of epochs, or by reducing the neural network size. The GBDT cost also can be reduced by lowering the number of weak estimators. The main reason for the expensive cost of GP is its computational complexity of $\mathcal{O}(N_{\text{train}}^3)$, and it can be improved by using scalable variants of the GP[25]. Also, note that all timing measurements are done with the ML implementations in the scikit-learn library. The cost comparison could be changed with the faster implementations dedicated for each ML algorithm, such as the PyTorch[26], XGBoost[27], and GPflow[28].

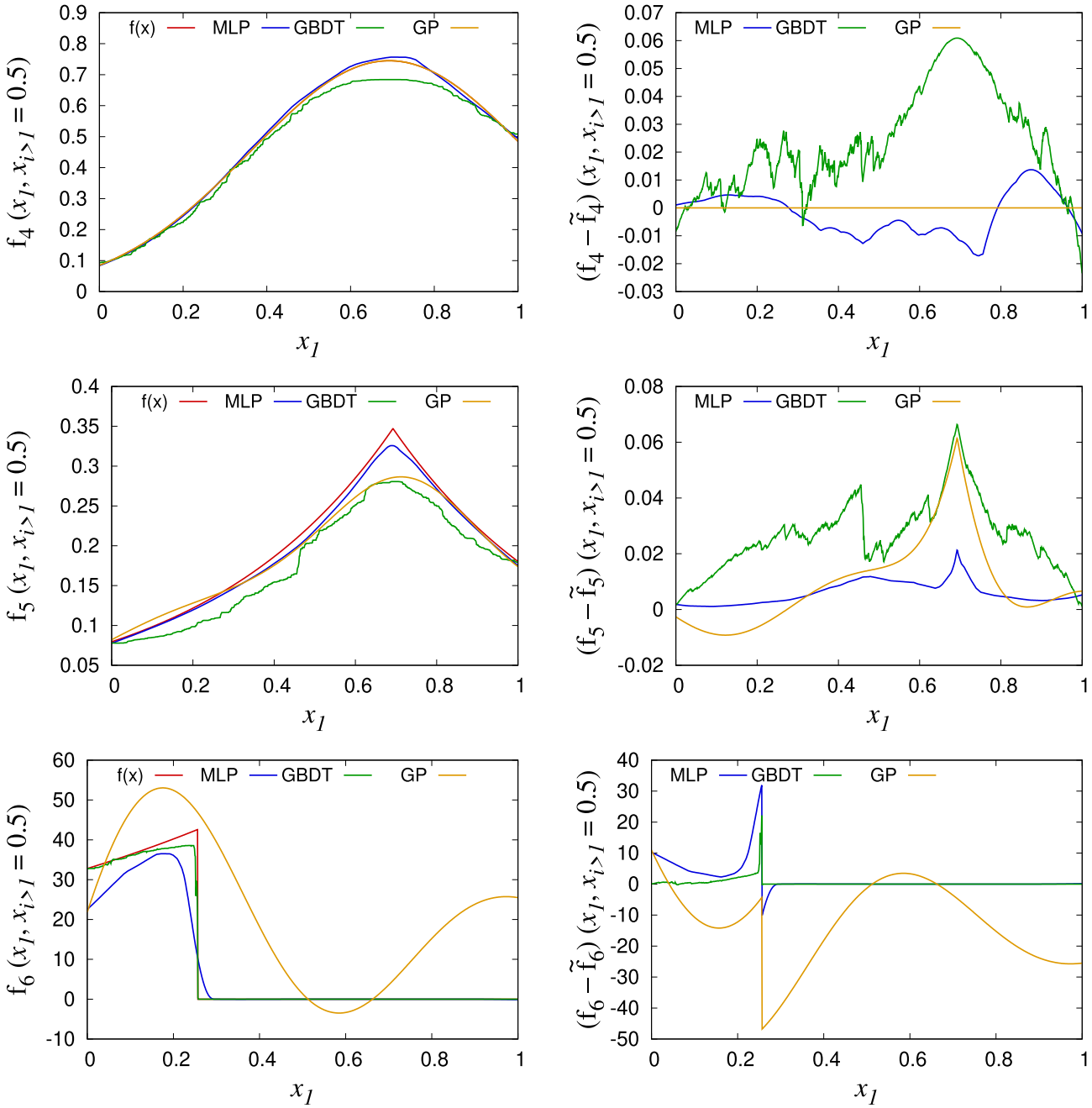


**Figure 1.** Integrands $f_i(\mathbf{x})$ and their ML predictions $\tilde{f}_i(\mathbf{x})$ (left), and the prediction errors $f_i(\mathbf{x}) - \tilde{f}_i(\mathbf{x})$ (right) for $N \approx 5000$, D=8, and $||\mathbf{c}||_1 = 8.0$. Those are plotted as a function of $x_1$, while rest of the $\mathbf{x}$ are fixed to $x_{i=2,3,\ldots,8} = 0.5$.

## Conclusion

In this paper, we proposed a novel algorithm calculating multi-dimensional integrals using ML regression algorithms. In this algorithm, a ML regression model is trained to mimic the target integrand and is used to estimate an approximated integral. Any bias of the estimate induced by the ML prediction error is corrected by using a bias correction term, as described in Eq. (2), so that the final integral result could have a statistically correct estimation of the uncertainty. Two essential prescriptions for obtaining a good the training efficiency are (1) collecting training samples using the VEGAS algorithm, and (2) scaling the training data using the nth-root scaling defined in Eq. (6).

The performance of the proposed ML integrator is compared with that of the VEGAS algorithm on six different integrand families listed in Eq. (8). Three ML regression algorithms of MLP, GBDT, and GP are examined, and the best performing algorithm is selected for each integrand family. For all test cases, the ML integrator shows better performance than the VEGAS for a given total number of integrand evaluations. In most of the cases, the

| Integrand family | | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Training | MLP | 31.8 (1.0) | 32.0 (1.0) | 40.97 (93) | 31.11 (81) | 48.0 (2.3) | 13.77 (82) |
| cost | GBDT | 3.235 (32) | 3.169 (24) | 4.276 (24) | 3.092 (21) | 3.248 (22) | 1.9468 (81) |
| (secs) | GP | 114.0 (6.7) | 111.8 (5.7) | 240.9 (9.8) | 102.3 (5.8) | 42.09 (78) | 13.39 (37) |
| Pred. | MLP | 0.5016 (27) | 0.5392 (42) | 0.5220 (38) | 0.5430 (43) | 0.5596 (49) | 0.5423 (24) |
| cost | GBDT | 2.554 (34) | 2.877 (21) | 2.674 (16) | 2.800 (12) | 2.967 (17) | 2.242 (46) |
| (secs) | GP | 9.412 (37) | 9.589 (63) | 14.65 (13) | 9.587 (45) | 8.25 (14) | 5.006 (89) |

**Table 2.** Wallclock time (s) spent for training (upper) and prediction on $10^5$ random samples (lower) for different ML algorithms with the integrand parameters of $N \approx 5000$, $D = 8$, and $||\mathbf{c}||_1 = 3.0$. $N_{\text{train}} = 2500$ for integrand families 1, 2, 4, and 5, $N_{\text{train}} = 3500$ for integrand family 4, and $N_{\text{train}} = 1500$ for integrand family 6. The results are averaged over 36 runs of random integrands as described in "Numerical experiments" section. The numbers in the parentheses are the standard deviation of the mean. The wallclock time is measured using a single core of an Intel Xeon E5-2695 v4 processor at 2.10GHz.

ML integrator is able to provide integration results with more than an order of magnitude smaller uncertainty than the VEGAS algorithm. The performance gain is presented in Table 1.

In this study, we compared the precision of the algorithms for a fixed number of integrand evaluations, ignoring the computational cost used for the ML training and predictions. Depending on the ML algorithm, when the ML cost is very expensive, the application of this algorithm could be limited to the problems whose integrand is expensive to evaluate, such as the problems reported in Ref.[6]. To make the algorithm applicable to a wider class of problems, a study of cost effective ML regression models will be needed in the future.

We find that the performance and the stability of the proposed algorithm largely depend on the smoothness of the regression output. Developing a ML algorithm specifically targeting the ML integrator will be able to improve the performance and stability of the algorithm. One possible approach is to augment the training data by adding a small amount of noise to the training data[29,30], which could improve the smoothness of the MLP and GBDT models. We also find that the GP regression algorithm with a RBF kernel fails in describing $C^0$ and discontinuous functions because of the singular points in the integrands. For a given integrand with known such singular points, one would be able to build a combination of multiple GP models defined on each domain divided by the singular points for a better performance. It will be also promising to explorer different types of kernels[31] and to develop a hybrid model of decision tree and GP that can be generically applicable for such integrands.

Received: 22 June 2021; Accepted: 3 September 2021
Published online: 23 September 2021

## References

1. Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. *Numerical Recipes 3rd Edition: The Art of Scientific Computing* 3rd edn. (Cambridge University Press, Cambridge, 2007).
2. Melnikov, K. & Petriello, F. *Phys. Rev. D* **74**, 114017. https://doi.org/10.1103/PhysRevD.74.114017 (2006). arXiv:hep-ph/0609070.
3. Gavin, R., Li, Y., Petriello, F. & Quackenbush, S. *Comput. Phys. Commun.* **182**, 2388. https://doi.org/10.1016/j.cpc.2011.06.008 (2011). arXiv:1011.3540 [hep-ph].
4. Denner, A. & Dittmaier, S. *Nucl. Phys. B* **734**, 62. https://doi.org/10.1016/j.nuclphysb.2005.11.007 (2006). arXiv:hep-ph/0509141.
5. Taruya, A., Nishimichi, T. & Bernardeau, F. *Phys. Rev. D* **87**, 083509. https://doi.org/10.1103/PhysRevD.87.083509 (2013). arXiv:1301.3624 [astro-ph.CO].
6. Sievert, M. D., Vitev, I. & Yoon, B. *Phys. Lett. B* **795**, 502. https://doi.org/10.1016/j.physletb.2019.06.019 (2019). arXiv:1903.06170 [hep-ph].
7. Peter Lepage, G. *J. Comput. Phys.* **27**, 192. https://doi.org/10.1016/0021-9991(78)90004-9 (1978).
8. Lepage, G. P. *VEGAS—an adaptive multi-dimensional integration program*, Technical Report CLNS-447 (Cornell University Laboratory of Nuclear Studies, Ithaca, NY, 1980). http://cds.cern.ch/record/123074
9. Lepage, G. P. *J. Comput. Phys.* **439**, 110386. https://doi.org/10.1016/j.jcp.2021.110386 (2021). arXiv:2009.05112 [physics.comp-ph].
10. Bendavid, J. (2017). arXiv:1707.00028 [hep-ph]
11. Bali, G. S., Collins, S. & Schafer, A. *Comput. Phys. Commun.* **181**, 1570. https://doi.org/10.1016/j.cpc.2010.05.008 (2010). arXiv:0910.3970 [hep-lat].
12. Blum, T., Izubuchi, T. & Shintani, E. *Phys. Rev. D* **88**, 094503. https://doi.org/10.1103/PhysRevD.88.094503 (2013). arXiv:1208.4349 [hep-lat].
13. Yoon, B., Bhattacharya, T. & Gupta, R. *Phys. Rev. D* **100**, 014504. https://doi.org/10.1103/PhysRevD.100.014504 (2019). arXiv:1807.05971 [hep-lat].
14. Rojas, R. *Neural Networks: A Systematic Introduction* (Springer, Berlin, 1996).
15. Breiman, L., Friedman, J., Stone, C., & Olshen, R. *Classification and Regression Trees*, The Wadsworth and Brooks-Cole statistics-probability series (Taylor & Francis, 1984). https://books.google.com/books?id=JwQx-WOmSyQC
16. Friedman, J. H. *Ann. Stat.* **29**, 1189. https://doi.org/10.1214/aos/1013203451 (2001).
17. Rasmussen, C. & Williams, C. *Gaussian Processes for Machine Learning, Adaptive Computation and Machine Learning* 248 (MIT Press, Cambridge, 2006).
18. Hahn, T. *Comput. Phys. Commun.* **168**, 78. https://doi.org/10.1016/j.cpc.2005.01.010 (2005). arXiv:hep-ph/0404043.
19. Genz, A. A package for testing multiple integration subroutines. In *Numerical Integration: Recent Developments, Software and Applications* (eds Keast, P. & Fairweather, G.) 337–340 (Springer, Netherlands, 1987). https://doi.org/10.1007/978-94-009-3889-2_33.
20. Pedregosa, F. *et al. J. Mach. Learn. Res.* **12**, 2825 (2011).

21. Kingma, D. P., & Ba, J. Adam: A method for stochastic optimization (2014). cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015 arXiv:1412.6980
22. Byrd, R., Lu, P., Nocedal, J. & Zhu, C. *SIAM J. Sci. Comput.* **16**, 1190. https://doi.org/10.1137/0916069 (1995).
23. Lepage, P. gplepage/vegas: vegas version 3.4.5 (2020a). https://doi.org/10.5281/zenodo.3897199
24. Lepage, P. vegas Documentation (2020b). https://vegas.readthedocs.io/en/latest/index.html
25. Liu, H., Ong, Y.-S., Shen, X., & Cai, J. When gaussian process meets big data: A review of scalable gps (2019). arXiv:1807.01065 [stat.ML]
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., L. Fang, Bai, J., & Chintala, S. In *Advances in Neural Information Processing Systems 32,* edited by Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. 8024–8035 (Curran Associates, Inc., 2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
27. Chen, T. & Guestrin, C. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16 ( ACM* 785–794 (New York, NY, USA, 2016).
28. Matthews, A.G. d.G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., & Hensman, J. GPflow: A Gaussian Process Library using TensorFlow. *J. Mach. Learn. Res.* **18**, 1. http://jmlr.org/papers/v18/16-537.html (2017).
29. Holmstrom, L. & Koistinen, P. *IEEE Trans. Neural Netw.* **3**, 24 (1992).
30. An, G. *Neural Comput* **8**, 643 (1996).
31. Schulz, E., Speekenbrink, M. & Krause, A. *J. Math. Psychol.* **85**, 1. https://doi.org/10.1016/j.jmp.2018.03.001 (2018).


## Acknowledgements

Computations were carried out using Institutional Computing at Los Alamos National Laboratory. This work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics under Contract No. 89233218CNA000001, and the LANL LDRD program.

## Author contributions

B.Y. designed the research, conducted numerical experiments, and wrote the manuscript.

## Competing interests

The author declares no competing interests.

## Additional information

**Supplementary Information** The online version contains supplementary material available at https://doi.org/10.1038/s41598-021-98392-z.

**Correspondence** and requests for materials should be addressed to B.Y.

**Reprints and permissions information** is available at www.nature.com/reprints.

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.