DOI: 10.1111/cgf.14341
Eurographics Symposium on Rendering 2021
A. Bousseau and M. McGuire
(Guest Editors)

Volume 40 (2021), Number 4

Q-NET: A Network for Low-dimensional Integrals of Neural Proxies

Kartic Subr, University of Edinburgh

Abstract

Integrals of multidimensional functions are often estimated by averaging function values at multiple locations. The use of an approximate surrogate or proxy *for the true function is useful if repeated evaluations are necessary. A proxy is even more useful if its own integral is known analytically and can be calculated practically. We design a family of fixed networks, which we call Q-NETs, that can calculate integrals of functions represented by sigmoidal universal approximators. Q-NETs operate on the parameters of the trained proxy and can calculate exact integrals over* any subset of dimensions *of the input domain. Q-NETs also facilitate convenient recalculation of integrals without resampling the integrand or retraining the proxy, under certain transformations to the input space. We highlight the benefits of this scheme for diverse rendering applications including inverse rendering, sampled procedural noise and visualization. Q-NETs are appealing in the following contexts: the dimensionality is low ($< 10D$); integrals of a sampled function need to be recalculated over different sub-domains; the estimation of integrals needs to be decoupled from the sampling strategy such as when sparse, adaptive sampling is used; marginal functions need to be known in functional form; or when powerful Single Instruction Multiple Data/Thread (SIMD/SIMT) pipelines are available.*

1. Introduction

The estimation of integrals is a computational bottleneck across several applications including rendering. In image synthesis, the radiant energy through each pixel is expressed as a multidimensional integral over the pixel surface, camera's aperture, exposure time, reflected directions, etc. The *integrand*—or function whose integral is sought— is a lightfield that spans space, angle and time. Various operations in the image formation process can be expressed as slices, projections or convolutions of the lightfield. The integrand is rarely available in closed form and is typically quite costly to evaluate. For example, it might require multi-bounce raytracing through a large and complex environment.

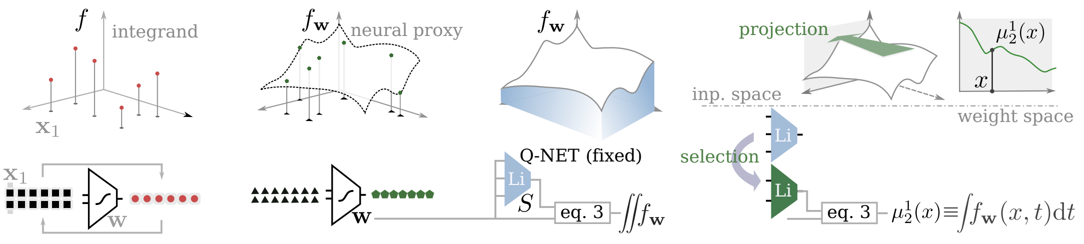
We explore the benefits of constructing an approximate integrand, or *proxy*, that is easier to evaluate. Although high-fidelity reconstruction of multidimensional functions is known to be challenging [HJW*08, KDBB17], numerical integration requires fewer samples than reconstruction [Dur11, SK13, RAMN12] to achieve the same mean-squared error (MSE). A faithful proxy is therefore also expected to be effective for numerical integration provided that a procedure is known to calculate its integral. Further, for rendering applications, it would be beneficial if the proxy accommodates transformations (such as those mentioned above) and if multiple projections can be calculated efficiently.

In this paper, we investigate a practical method to estimate integrals and marginals of a class of neural network proxies. Shallow feed-forward networks (SFFN) consist of one hidden layer with a sigmoid activation function and a purely linear output layer. This textbook case is an example of a *universal approximator network* [Cyb89, Hor91, LPW*17]. It can approximate [SCC15] and integrate [LIA20] any continuous function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ accurately. During training, samples of f are used to learn parameters $\mathbf{w}$ of the network so that it represents the functional approximation $f_\mathbf{w} \approx f$ which can be evaluated rapidly anywhere in the domain. The simplicity and universality of sigmoidal approximators make them an attractive first choice as neural proxies for numerical integration. We present Q-NETs as a general and practical tool for numerical integration across a variety of rendering applications.

Given a trained proxy $f_\mathbf{w}$, we design a family of shallow neural networks (Q-NETs) to evaluate the exact functional marginals of $f_\mathbf{w}$ in closed form by integrating over any subset of the input space $\mathbb{R}^d$. These marginals can be estimated using $\mathbf{w}$ without further sampling of f. The proxy $f_\mathbf{w}$ is also useful (Sec. 4) as a control variate [Owe13, Sec. 8.9]. Q-NETs simplify integral calculations and can accommodate operations such as projection (see Fig. 1) without the need for resampling f or retraining $f_\mathbf{w}$. Our approach is practical since standard implementations of SFFN may be leveraged for fast computation on graphics processing units.

Contributions:.

- we design fixed-weight SFFNs to calculate integrals and marginals of functions represented by sigmoidal SFFN (Sec. 3.1);
- we derive transformations to the inputs and network weights to act as counterparts to operations on the input space (Sec. 3.2);
- we assess the empirical fidelity of integration via Q-NETs for functions with discontinuities (Sec. 4) ;
- we demonstrate its versatility across problems such as inverse rendering, sampling of procedural noise visualization (Sec. 5).



(a) training proxy (b) evaluating proxy (c) calculating integrals (d) calculating functional marginals

INP. SPACE	WT. SPACE
affine trans.	matrix mult.
summing	concatentation
slicing	column select
projection	selection
sub-domain	modify S

(e) other operators

Figure 1: *Overview. (a) Sampled function f. (b) Regression via training a 1-layer sigmoidal universal approximator yields a proxy function $f_\mathbf{w}$. (c) Our fixed network (no learnable parameters) which we call a Q-NET operates on network parameters $\mathbf{w}$. (d) Any marginal (projection) of $f_\mathbf{w}$ can be obtained in functional form via an input selection transformation to Q-NETs. (e) A list of other transformations to $f_\mathbf{w}$ which can be accommodated via modification of network parameters.*

2. Background

2.1. Related work

Numerical integration (general). Quadrature schemes [BP11, KKN18] approximate definite integrals by dividing the integration domain into cells (usually uniformly along each dimension), approximating the function using polynomials within each cell and summing up the analytically computed integrals within each cell. As the dimensionality of the domain increases, the number of cells and hence number of samples required increases exponentially. This is referred to as the 'curse of dimensionality'. Monte Carlo (MC), Quasi-MC (QMC) and Markov Chain MC (MCMC) operate differently, by expressing integrals as expectations which can be estimated via simulation [MU49]. The simulation is (pseduo) random for MC and MCMC. These methods converge slowly – MC converges at $O(1/\sqrt{N})$. Several variants [Owe13] address the slow convergence by striking a compromise between *bias* (accuracy) and *variance* (precision). QMC methods [Nie78, Nie92] replace stochasticity with carefully designed, deterministic samples which improves convergence dramatically when integrands are smooth and integration is over moderate dimensionalities.

Computer Graphics adaptations. Quadrature schemes have been used for antialiasing [GT96], to render participating media [PH89, JLSJ11, FWKH17], for discretized time-integration of Laplacians physics-based animation [KYT*06] and subspace deformation [AKJ08]. MC (or QMC) path tracing [Kaj86] and its MCMC variants [Vea98] form the industry standard [FHH*19, CJ16] for estimating multidimensional integrals in offline rendering applications. MC and MCMC methods have been honed for rendering via analyses in Fourier [DHS*05, BSS*13], wavelet [ODR09] and gradient-domains [LKL*13, KMA*15]. Recently, a neural control variate [MRKN20] that was tailored to light transport estimation produced an impressive reduction in variance. The discretization of time for physics based animation necessitates a different class of integrators [BS19, WLF*20] which reformulate differential systems variationally and solve time integration as an optimization problem.

Machine Learning applications. Bayesian methods routinely use probabilistic model-based surrogates for expensive integrands [GR03] or to guide active sampling (adaptive Bayesian quadrature [OGG*12, KH19], Bayesian Optimization [SSW*16], etc.). This approach can be used to estimate the marginal likelihood [BOGO15, GOG*14], approximate the posterior [KSP15, WL18] and to simultaneously infer both [Ace18]. They operate by imposing a Gaussian Process (GP) prior on the integrand and using analytical formulae for the expectation and uncertainty of the statistical surrogate. Hybrid probabilistic neural networks use GPs to model neural weights [KB20] for calibrated reasoning about uncertainty. *Integral representations* [PDW20] use an analysis of continuous distributions, for a particular target function, from which instances of shallow neural networks can be sampled. Deep neural networks can mimic MC solutions to certain partial differential equations [GJS19], sidestepping the curse of dimensionality.

Integration using Neural Networks. Integration formulae have been derived for shallow networks with different activation functions in 1D [ZZYNH06, YDW13] and hyperrectangular domains [LIA20, Sub20]. The formulae are unwieldy and expensive to evaluate. Alternatively, the derivatives of a neural network have been trained to match the integrand [TNVdVG19, LMW20] so that the trained neural network evaluates the integral. This enables the use of deeper networks but marginalization is only possible in the (reverse) sequence of variables used during training. A third approach uses integral-preserving neural layers (flows) to map a general distribution (such as a Gaussian) onto a proxy for integrands in light transport simulation [MRKN20]. The proxy was used as a control variate along with an importance sampling scheme for the residual. Our method provides a proxy, allows flexible marginalization, enables transformations and can be evaluated efficiently (<10D) but it is limited to one hidden layer (Table 1). Generally, definite integrals are obtained by accumulating (appropriate signs) integrals. e.g. if Γ is a 2D cdf, the definite integral over $[0,1]\times[0,1]$ is given by $\Gamma(1,1)-\Gamma(1,0)-\Gamma(0,1)+\Gamma(0,0)$. This procedure is common to methods over hyperrectangular domains (including ours) and usually dominates the overall computational cost in high dimensions.

method	*proxy*	*marginals*	*transf.*	*speed*	*depth*
[LIA20]	🟢	🟢	○	🔴	🔴
[LMW20]	🔴	○	🔴	🟢	🟢
[MRKN20]	🟢	🔴	🔴	🟢	🟢
Ours	🟢	🟢	🟢	🟢	🔴

Table 1: *Comparison of recent and relevant work.*

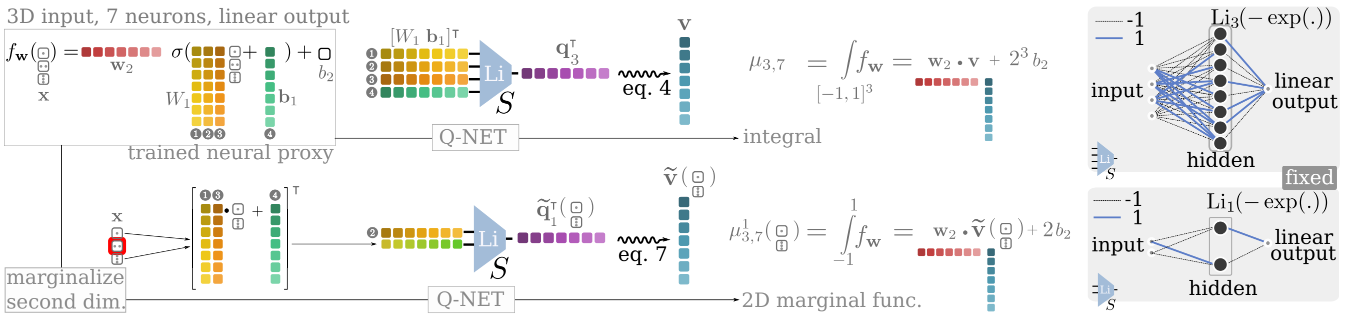


Figure 2: *Given parameters* $\mathbf{w} \equiv (W_1, \mathbf{w}_2, \mathbf{b}_1, b_2)$ *of a sigmoidal approximator for a function in 3D, the integral of the function (top half) and its projection (bottom half) are calculated similarly but with different instances of Q-NETs (blue trapezoids). In the latter case, the 2D marginal is a function of the first and third input variables. Q-NETs are fixed networks (shown on the right) that depend only on the number of dimensions being integrated.*

2.2. Notation and formula

We denote row and column vectors with boldface characters (e.g. $\mathbf{x}$, $\mathbf{b}_1$, $\mathbf{w}_2$) and matrices using capital letters (e.g. W_1). We use superscripts to select elements of a vector or matrix. e.g. $\mathbf{b}_1^i$ and $W_1^{i,\cdot}$ represent the i^{th} element $\mathbf{b}_1$ and i^{th} row of W_1 respectively. Without loss of generality (see Sec. 6) we assume a normalized hyper-rectangular domain $\mathbf{x} \in \mathcal{D} \equiv [-1,1]^d$. We approximate the function $f : \mathcal{D} \to \mathbb{R}$ with $f_\mathbf{w} : \mathcal{D}^d \to \mathbb{R}$ obtained by a training a shallow feedforward neural network with one hidden layer (k neurons) and a linear output layer using N samples $f(\mathbf{x}_n)$ $n = 1, \cdots, N$. $\mathbf{w}$ collectively encodes all learnable parameters of the network: a $k \times d$ matrix W_1, a k-dimensional row vector $\mathbf{w}_2$, a k-dimensional column vector $\mathbf{b}_1$ and a real number b_2 so that

$$f_\mathbf{w}(\mathbf{x}) = \mathbf{w}_2 \;\; \sigma(W_1\mathbf{x} + \mathbf{b}_1) + b_2, \text{ where } \sigma^i(\mathbf{z}^i) \equiv \frac{1}{1+e^{-\mathbf{z}^i}}. \quad (1)$$

That is, σ operates independently on each of the k elements. We design a network with fixed weights to calculate $\mu_{d,k}(\mathbf{w})$, the integral of $f_\mathbf{w}$ over $\mathcal{D}$ exactly. Thus $\mu_{d,k}(\mathbf{w})$ estimates the integral of f:

$$\mu_{d,k}(\mathbf{w}) \equiv \int_\mathcal{D} f_\mathbf{w}(\mathbf{x})\, \mathrm{d}\mathbf{x} \;\approx\; I \;\text{ where }\; I \equiv \int_\mathcal{D} f(\mathbf{x})\, \mathrm{d}\mathbf{x}. \quad (2)$$

We set $k \propto N^{\frac{1}{1+d}}$ causing μ to be a consistent estimator for I. In practice, since k is fixed (finite), this estimator is not consistent but our experiments validate its utility nevertheless.

The integral of a sum of shifted and scaled sigmoids is a weighted sum of the integrals of sigmoids. The logistic sigmoid and its integrals are instances of a special function called the polylogarithm [Eul68, Lew81] which can be evaluated efficiently [Cra06]. The formula for the integral was derived in concurrent work [LIA20, Sub20] as:

$$\mu_{d,k}(\mathbf{w}) = \mathbf{w}_2 \;\mathbf{v}\; + \;2^d\; b_2, \quad (3)$$

$$\text{where } \mathbf{v}^i = 2^d \;+\; \frac{1}{\tilde{\mathbf{w}}_1^i} \sum_{m=1}^{2^d} \alpha_m \,\mathrm{Li}_d\left(-\exp(S^{m,\cdot}\; W_1^{i,\cdot\;\top} - \mathbf{b}_1^i)\right).$$

Here $\mathbf{v}$ is a column vector representing integrals of each of the neurons in the hidden layer and $\mathrm{Li}_d(x)$ is the polylogarithm function of order d. The summation is due to the definite integral requiring appropriate addition or subtraction of the integral at each of the vertices of the hypercube. The 2^d vertices (rows) are represented by S, whose elements are ± 1. The contribution at each vertex is positive ($\alpha_m = 1$) when there are an even number of -1s in the row $S^{m,:}$ and negative ($\alpha_m = -1$) otherwise. The division by $\tilde{\mathbf{w}}_1^i \equiv \prod_j W_1^{i,j}$, the product of the elements of the i^{th} row of W_1, arises due to the integration of transformed sigmoids.

3. Q-NETs

Once a proxy $f_\mathbf{w}$ is trained, we design Q-NETs for practical calculation of marginals of $f_\mathbf{w}$. Another advantage is that some transformations to the input domain may be accommodated elegantly without the need for re-sampling f or retraining $f_\mathbf{w}$.

3.1. Central observation

The calculation of the elements of $\mathbf{v}$ in Eq. 3 can be simplified by using a shallow feedforward network q_d if they are reformulated as

$$\mathbf{v}^i = 2^d + \frac{q_d(\mathbf{y}_i)}{\tilde{\mathbf{w}}_1^i} \quad (4)$$

$$\text{where} \qquad q_d(\mathbf{y}_i) \equiv \mathbf{w}_3\; \sigma_d\left(\left[S \;\; -\mathbb{1}_{2^d}\right] \mathbf{y}_i\right). \quad (5)$$

Here $\mathbf{y}_i \equiv [W_1^{i,\cdot} \;\; \mathbf{b}_1^i]^\top$, $\sigma_d(.) \equiv \mathrm{Li}_d(-\exp(.))$ and $\mathbb{1}_{2^d}$ is a column of 2^d ones. The advantage of this representation is that Eq. 5 is similar in structure to Eq. 1 and can therefore be computed using a feedforward network with fixed weights (no learnable parameters), input $\mathbf{y}_i$, output $q_d(\mathbf{y}_i)$, one hidden layer containing 2^d neurons and the activation function σ_d. The biases of this network are zero and its input and output weights are $\left[S \;\; -\mathbb{1}_{2^d}\right]$ and $\mathbf{w}_3$ respectively. $\mathbf{w}_3$ is a row vector whose m^{th} element is α_m. We call this family of networks, parameterized by d, Q-NETs since it enables quadrature of the approximator network. In practice, all k vectors may be stacked (as columns) into a matrix $[W_1 \; \mathbf{b}_1]^\top$ for efficient (vectorized) evaluation. In this case $\mathbf{w}_3$ will be replaced by a matrix each of whose k

rows is a vector $\mathbf{w}_3$. See the top row of Fig. 2 for an example of the role of a Q-NET in calculating integrals. Q-NETs form an elegant representation with interpretable properties.

3.2. Operations on Q-NETs

We highlight key properties (also see the accompanying video) of Q-NETs which we exploit in the applications shown in Sec. 5.

Affine transformation. If the input space is transformed as $M\mathbf{x}+c$ where M is a transformation matrix and c is a translation,

$$\begin{aligned}\tilde{f}_{\mathbf{w}}(\mathbf{x}) &= \mathbf{w}_2\ \sigma(W_1\ (M\,\mathbf{x}+c)+\mathbf{b}_1)+b_2\\ &= \mathbf{w}_2\ \sigma(\tilde{W}_1\,\mathbf{x}+\tilde{\mathbf{b}}_1)+b_2,\end{aligned} \tag{6}$$

where $\tilde{W}_1 \equiv W_1 M$ and $\tilde{\mathbf{b}}_1 \equiv \mathbf{b}_1 + W_1 c$. The integral of the transformed function $\tilde{f}_{\mathbf{w}}(\mathbf{x})$ may be calculated just as for f, but with the modified weights $[\tilde{W}_1^{i,\cdot}\ \ \tilde{\mathbf{b}}_1^i]^\top$ in Eq. 6 as inputs to the Q-NET and then divided by the absolute value of the determinant of the Jacobian of the affine transformation. Integrals over non-axis-aligned hyperrectangles may be computed using this property.

Projection. To marginalize $r<d$ input dimensions of $\mathbf{x}$, the variables $\mathbf{x}^{1,\cdots,r}$ need to be integrated (we assume without loss of generality that the first r dimensions are marginalized), yielding a function in the remaining variables $\mu^r_{d,k}(\mathbf{w},\mathbf{x}^{r+1,\cdots,d})$. Using Q-NETs, the procedure is to use $\tilde{\mathbf{v}}^i$, instead of $\mathbf{v}^i$ as for the full integral, where

$$\tilde{\mathbf{v}}^i \ \equiv\ 2^r + q_r(\tilde{\mathbf{y}}_i)\ /\ \prod_{j=1}^{r} W_1^{i,j}. \tag{7}$$

Compared to Eq. 5 d has been replaced with r on the rhs and the input to the Q-NET is $\tilde{\mathbf{y}}_i \equiv [W_1^{i,1,\cdots,r}\ \ (\mathbf{b}_1^i + W_1^{i,r+1,\cdots,d}\mathbf{x}^{r+1,\cdots,d})]^\top$ instead of $\mathbf{y}_i$. Thus projection along a subspace of $\mathbf{x}$ amounts to a *selection* operation in $\mathbf{y}_i$ where the weighted non-marginalized variables are moved from being individual inputs to Q-NET to an aggregate input along with the last (bias) dimension of $\mathbf{y}_i$.

Slicing. If the proxy is sliced through $r<d$ dimensions using the constants $\mathbf{x}^{1,\cdots,r} = c^{1,\cdots,r}$, the resulting $(d-r)$ dim. function is

$$\tilde{f}_{\mathbf{w}}(\mathbf{x}^{r+1,\cdots,d}) = \mathbf{w}_2\ \sigma\left(\tilde{W}_1\ \mathbf{x}^{r+1,\cdots,d} + \tilde{\mathbf{b}}_1\right)+b_2, \tag{8}$$

where $\tilde{W}_1 \equiv W_1^{\cdot,(r+1,\cdots,d)}$ and $\tilde{\mathbf{b}}_1 \equiv \mathbf{b}_1 + W_1^{\cdot,(1,\cdots,r)} c^{1,\cdots,r}$. Slicing along subdimensions of $\mathbf{x}$ amounts to removing the corresponding columns of W_1 and adding the product of those columns with the slicing constants to the bias $\mathbf{b}_1$. Again, since the sliced function can be obtained via manipulation of the original weights, the integral of the sliced function may be calculated just as before via a Q-NET but with the following two changes: 1) the dimensionality of integration is $d-r$ instead of d; and 2) the input to the Q-NET is $[\tilde{W}_1^{i,\cdot}\ \ \tilde{\mathbf{b}}_1^i]^\top$ with the weight matrix and bias vector from Eq. 8.

Integrals over sub-domains. change of domain may be achieved via scaling and translation. Alternatively, it may be realised by changing S. We describe the latter, since it hints at the future possibility of inferring the domain (learning S) given an integral. The domain $[-1,1]^d$ manifests in the Q-NET formulation in two ways: the constant 2^d is the product of the differences between upper and lower limits in each dimension; and matrix S contains ± 1 to represent the vertices of the hyperrectangle. To modify the integration limits to $\mathbf{a},\mathbf{b} \in [-1,1]^d$, first 2^d needs to be replaced by $\prod(\mathbf{b}^j - \mathbf{a}^j)$ in Eq. 3 and Eq. 5 (j iterates through d dimensions). Then, the rows of S must be updated with combinations of elements of $\mathbf{a}$ and $\mathbf{b}$ to list vertices of the hypercube defined by the new limits.

3.3. Complexity and error

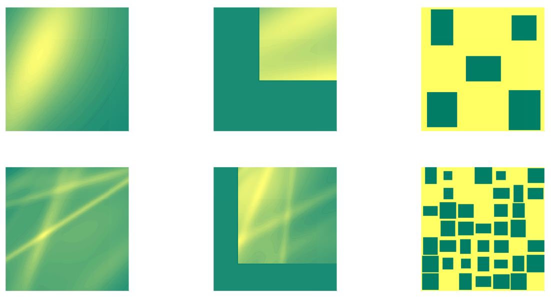
The computational complexity of evaluating the formula directly is $O(k\,d\,2^d)$. Using a Q-NET facilitates parallel and vectorized computation across neurons, across dimensions or the 2^d rows of the sign matrix as necessary. Given a trained proxy, computation time is independent of N, the number of samples. However, the memory complexity is $O(kd)$ for direct evaluation (no need to store S) compared to $O(kd+d^2)$ with Q-NETs via binary encoding of S which occupies $O(d)$ space. Since d is the integrated dimensions, marginalizing along a small number of dimensions remains feasible even when the domain of the integrand is high-dimensional.

Theorem 1 The upper bound for the squared error between integrals I and μ of a c-times differentiable function $f:\mathbb{R}^d \rightarrow \mathbb{R}$ and its shallow sigmoidal approximant $f_{\mathbf{w}}$ (with k neurons) is given by

$$(I-\mu)^2 \ <\ \frac{\tilde{h}}{k^{2c/d}} \ -\ \mathrm{V}[f-f_{\mathbf{w}}]. \tag{9}$$

Here $\mathrm{V}[.]$ denotes the variance operator and $\tilde{h}$ is the first moment of the Fourier spectrum of f. Please refer to the Appendix for a proof.

The variance term on the rhs of Eq. 9 tightens the bound compared to reconstruction error. The change of metric from one that measures reconstruction fidelity of the proxy to its integration fidelity results in a lower upper bound. This is qualitatively consistent with previous analyses of numerical integration of bandlimited signals [Dur11, SK13, RAMN12] which also conclude that integration is relatively more sample-efficient. Since Theorem 1 assumes smoothness, we first perform quantitative validation of Q-NETs on integrands (see Fig. 3) with discontinuities.



a) GM b) DGM c) HR

Figure 3: *Test integrands for empirical validation: (a) Gaussian mixtures (GM), (b) GM with d discontinuities (GMD) and (c) binary hyperrectangles (HR). Two different parametric settings are shown (rows) for each family.*

4. Results I: Experimental validation

We use three classes of integrands for empirical tests: Smooth integrands represented by Gaussian mixtures (GM), integrands with d

discontinuities (GMD) and arbitrarily discontinuous integrands using sums of binary hyperrectangles (HR). We averaged results for different random parameters of these functions in the domain $[0,1]^d$ for dimensions $d \leq 12$. Fig. 3 visualizes each class (columns) in 2D for two different choices of parameters (rows). We ran experiments in MATLAB on a Desktop with an Intel 8-Core i7-6700 processor, 32 GB of RAM and an NVIDIA TITAN RTX GPU. Training times depend on f, d and k, but integration using Q-NETs was typically at rates of about 1000, 550 and 300 neurons/second per thread for 1D, 2D, and 3D integrals respectively.

Marginalization. The projection operator uses Q-NETs defined over a subset of dimensions. Fig. 4 shows an assessment of integrating a 3D Gaussian mixture with 35 components using $k = 35$ neurons and $N = 2048$ samples. The figure omits subscripts on μ for brevity. The point-sampled functions μ^1 and μ^2 are plotted for different permutations of the components of $\mathbf{x}$. The marginals obtained using Q-NETs match references for the projections. We demonstrate its application to Bayesian inverse rendering in Sec. 5.1.

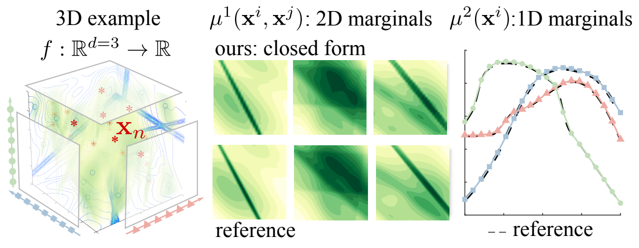


Figure 4: *Marginals of a 3D Gaussian mixture (left) along one dimension (middle) and two dimensions (right). The resulting 2D and 1D marginals, evaluated on grids, are compared with references.*

Sub-integrals. We trained $f_\mathbf{w}$ using $k = 180$ neurons and N samples of the zone plate function $f(\mathbf{x}) = (1+\cos(220\,\mathbf{x}^\mathsf{T}\mathbf{x}))/2$ in $[0,1] \times [0,1]$ and estimated integrals over subdomains of different sizes. Fig. 5 visualizes f (left), the subdomain sizes (white boxes) and plots of relative root mean squared error (RRMSE) on the right as N is increased up to 30K. The plot shows the mean RRMSE along with the standard deviations (shaded region) over 100 randomly shifted square subdomains with sides $1/3$, $1/8$ and $1/20$. Average error is larger for smaller subdomains, as expected, since the number of expected samples representing the function within the subdomain drops quadratically with respect to the side. That is, for this function, integrals over arbitrary sub-domains as small as $1/20 \times 1/20$ can be obtained with 10% error if $f_\mathbf{w}$ was trained using a 170×170 grid over the unit square. Error curves for integration over the entire domain ('1' in the legend) using Q-NET (solid) and QMC (dotted) are also shown.

Increasing dimensionality. The plots in Fig. 7 confirm that our estimate (red) is consistent, improves upon general Monte Carlo integration and is competitive with Quasi-Monte Carlo methods in 2D. However in 5D the proxy seems less effective when discontinuities are present (e.g. HR). We investigated this further by measuring relative errors and variances for dimensions up to $d = 12$ using HR integrands. Fig. 6 plots these for different N (colors) and two choices for k, the number of neurons (solid and dashed lines) using a log

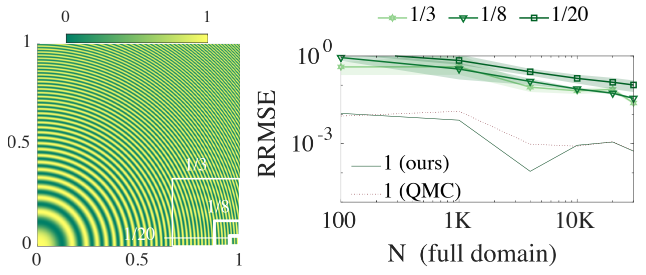


Figure 5: *We tested our method for recalculating integrals across sub-domains (without retraining) by training $f_\mathbf{w}$ on the zone plate (left) function $(1+\cos(220x^2+220y^2))/2$ using N samples in the unit square. The plot shows RRMSE (green curves) of 100 random, square subdomains of sizes $1/3$, $1/8$ and $1/20$ as N is increased. The errors for integrating over the whole domain is shown (black).*

scale for error and a linear scale for dimensions. Although these plots confirm the increase in error from $d = 2$ to $d = 5$, they also provide reassurance that the increase flattens down towards $d = 12$.

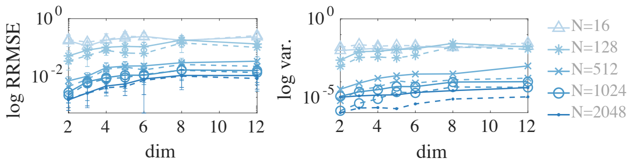


Figure 6: *Increasing dimensionality on HR integrands. Plots of relative error (left) and relative variance (right) averaged over 40 iterations each of 50 random HR integrands over dimensions $d =$ 2, 3, 4, 5, 6, 8, 12. Error increases with dimension but the effect is not sustained. Figure 7 shows convergence plots for 2D and 5D.*

Direct use for integration. We trained $f_\mathbf{w}$ using N samples from each randomly generated instance of integrand f. Then we evaluated a Q-NET using the trained parameters and calculated relative error against analytical references. Fig. 7 plots the mean convergence of RRMSE for each class (columns) of integrands for $d = 2$ (top row) and $d = 5$ (bottom row). Trend lines (dashed) on the log-log plots depict $O(N^{-0.5})$ and $O(N^{-1.5})$ rates of convergence while error bars indicate standard deviation across 250 integrands. The plots indicate that Q-NETs yield useful approximations even when used directly in low dimensions or with smooth functions. They also show that Q-NETs are suited to integration with large N (hence k). For discontinuous integrands they are useful if the proxy is used as a control variate.

Use as a control variate. We devised a family of estimators CV-Q-NET that use $f_\mathbf{w}$ as a control variate [Owe13, Sec. 8.9] to integrate f. Given $\nu \in [0,1]$, CV-Q-NET uses ceil$[(1-\nu)N]$ samples to train $f_\mathbf{w}$ and the remaining samples to integrate $f_\Delta(\mathbf{x}) = f(\mathbf{x}) - f_\mathbf{w}(\mathbf{x})$ via standard MC (or QMC). The final estimator is then the sum of the closed-form integral of $f_\mathbf{w}$ and the MC (or QMC) estimator. When $\nu = 0$, CV-Q-NET is equivalent to μ (Q-NET) and as ν tends to one it approaches pure MC (or QMC). Although unbiased, this still yields some error for a fixed k (see Fig. 8) but it results in

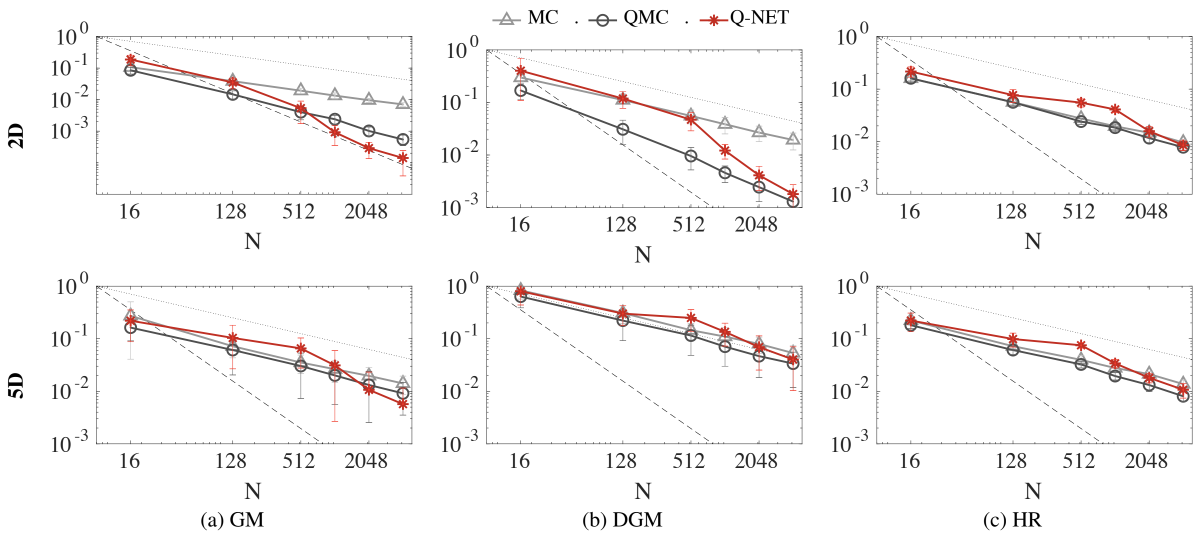


Figure 7: *Convergence plots (RRMSE vs N) up to 4K samples in 2D (top) and 5D (bottom) for three integrands families (columns). Q-NETs yield useful estimates but are not competitive with QMC when applied directly (without use as a control variate) to discontinuous functions. The convergence rate is curiously better than MC for $N > 512$ in all cases except HR. Dashed guide lines $O(N^{-1})$ and $O(N^{-1.5})$ are shown to facilitate comparisons of convergence rates. Figure 8 further investigates discontinuous integrands in 4D and 8D.*

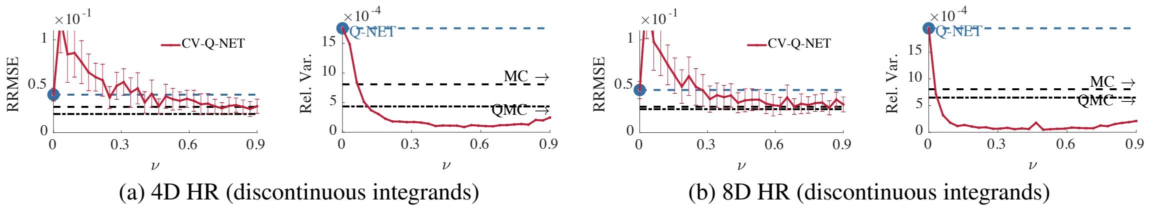


Figure 8: *The proxy is useful as a control variate (CV) by using a fraction (ν) of the samples to integrate the difference $f - f_{\mathbf{w}}$. The plots compare relative errors and variances of CV-QNET (red), Q-NET ($\nu = 0$) and MC and QMC ($\nu = 1$) estimators. CV-Q-NET is effective at reducing variance (lower than MC and QMC) for a wide range of values of ν in 4D as well as in 8D.*

variance reduction for a wide range of ν even in high-dimensional discontinuous functions.

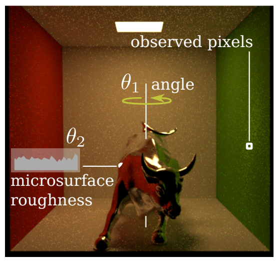
5. Results II: Sample applications

We demonstrate the utility of the proposed proxy and its integration via Q-NETs within a few computer graphics contexts. Our aim is to highlight the versatility of the proxy and its potential to inspire future work, rather than to claim improvement over state of the art in a specific application.

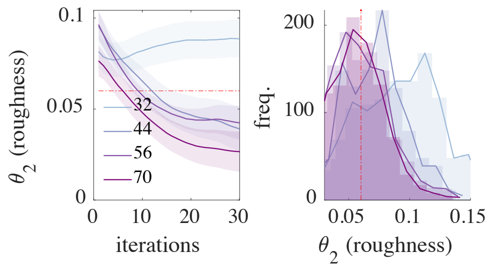
5.1. Bayesian inverse rendering

Consider the problem of estimating probability distributions over scene parameters θ(material, object transformations, etc.) given noisy, rendered observations of a small patch (6×6) of pixels. Excellent options are available [LHK*20, BLD20, NDVZJ19, ZWZ*19] for inferring point estimates of θ given some observed (target) radiance distribution ℓ. Typically these methods use a differentiable rendering pipeline enabling the optimization of θ via iterative back-propagation of gradients with respect to θ. These methods cope with impressively high-dimensional θ and do not require any prior knowledge or precomputation. They do however rely on a (heavily) modified differentiable rendering pipeline. It is possible that ℓ is not differentiable with repect to subsets of parameters. e.g. rotation transformations in Mitsuba 2 (θ_1 in Fig. 9a).

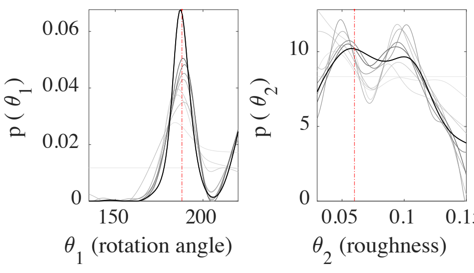
Inferring distributions over render parameters is challenging for existing differentiable renderers— even for the simple case of a single (and differentiable) parameter. Bayesian approaches, on the other hand, naturally model distributions in observed data and are popular for solving inverse problems [Stu10, CNN20]. We use precomputation from a standard forward renderer to infer the posterior distribution over render parameters θ given observed radiance ℓ_o. During precomputation, we render images using random vectors of parameters θ_i and record the radiance ℓ_i. We then train a neural proxy $f_{\mathbf{w}}(\theta, \ell)$ using (θ_i, ℓ_i) and using $k = 100$ neurons. We normalize this proxy (using a Q-NET) so that it approximates the joint probability distribution $p(\theta, \ell)$. At test time, given observed radiance values ℓ_o we iteratively perform Bayesian inference to obtain the posterior $p(\theta|\ell_o) = p(\ell_o|\theta)p(\theta)/p(\ell)$ starting with an initial uniform prior $p(\theta)$. At each iteration, we train $f_{\mathbf{w}} \approx p(\ell_o|\theta)p(\theta)$



(a) scene and parameters



(b) Point estimates for θ$_2$ using Mitsuba 2



(c) Bayesian inference using Q-NET

Figure 9: *(a) Given observed radiance (patch on the green wall) modern differentiable renderers like Mitsuba 2 [NDVZJ19] are effective at inferring point estimates for differentiable scene parameters such as* θ$_2$*. (b) We ran several iterations, using different patch-sizes for observations and plotted these estimates and their frequency polygons (histograms). (c) Our method can be used to infer distributions over parameters which may* (θ$_2$) *or may not* (θ$_1$) *be differentiable. We achieve this by using precomputed radiance samples from a standard forward renderer to train* $f_\mathbf{w}$ *to be the 3D joint distribution* $f_\mathbf{w}(\theta,\ell)$*. Then we perform Bayesian inference starting with a uniform prior (light grey). Given unseen observations (on the green wall), the iteratively refined posterior distributions over* θ$_1$ *and* θ$_2$ *are shown with progressively darker greys and reference values are shown with dashed red lines. (Also explained in the video)*

and update the prior to be the posterior from the previous iteration. The numerator and denominator of the Bayesian update are the marginals of $f_\mathbf{w}$ and are calculated using Q-NETs. Fig. 9.a shows an example scene where two parameters of the bull's model were varied: the angle of rotation (θ$_1$) around the vertical axis and its material roughness parameter (θ$_2$). The updated posterior marginals over θ$_1$ and θ$_2$ are evaluated on (1D) grids and plotted in Fig. 9.c.

We used a state of the art inverse renderer [NDVZJ19] to obtain estimates for θ$_2$ given observations ℓ_o on the green wall. The plots in Fig. 9.b (left) show point estimates vs iterations for different sizes of observed patches. Their histograms hint at the underlying distributions. Fig. 9.c plots iterative distributions (progressively darker grey) using our Bayesian inference over each parameter. Dashed red lines show the reference values used to generate the observed patches ℓ_o. We trained $f_\mathbf{w}$ using 100 simulations of a 6×6 crop on the green wall with 100 different θ$_i$. The training time with 50 and 150 neurons is 6 (8) and 10 (54) seconds with (without) GPU computation.

After 20 iterations, the modes of the inferred (black) distributions are close to the reference. Also, the results suggest a low confidence in the inferred θ$_2$, which is non-trivial to obtain robustly from point estimates. Although our inference scheme does not require a differentiable renderer, it relies on precomputation for learning. An animated summary is presented in the accompanying video.

5.2. Modeling with neural noise

Various classes of procedural noise are useful in modeling virtual worlds. Perlin noise [PH89], a type of lattice-gradient noise, is the de-facto choice due to its visual appeal, easy and efficient implementation and extensibility. Its computational cost is $O(d2^d)$ per evaluation for noise in d dimensions. Simplex noise [Per02] improves on this, with a cost of $O(d^2)$ per evaluation. Gabor noise is a popular alternative that can be trained from examples [GLLD12] and filtered [LD11]. Neural noise can be computed in $O(dk)$ per evaluation if k neurons are used. Our MATLAB implementation's speed is about 13 KHz, 30 KHz and 500 KHz for 25K, 10K and 100 neurons respectively (averaged over 1 million 2D evaluations).

Sigmoidal approximator networks with random **w** (no training) are useful generators of noise. They can be evaluated easily on GPUs, trained to resemble examples, sliced and transformed using intuitive parameter settings. Their integral enables exact filtering, normalization and marginalization. Q-NETs allow the noise to be sampled without *any* evaluations of the noise function. The bottom

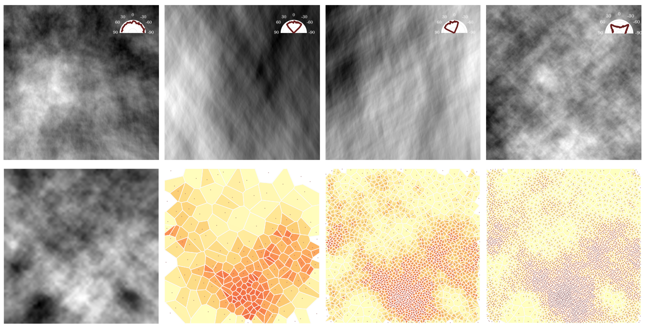


Figure 10: Top row: *Examples of noise with target anisotropies (insets).* Bottom row: *input 2D noise and samples generated (180, 1500 and 4500 samples).*

row in Fig. 10 visualizes a target noise pattern and 180, 1500 and 4500 Halton samples respectively, along with their Voronoi cells. The samples (best viewed by magnifying on a screen) were generated via proportional allocation over gridded tiles with the number of samples in each tile proportional to the integral of the noise within that tile (calculated using a Q-NET). The cells may also be generated via a k-d tree type construction where partitioning values at each step are the medians of the respective functional marginals obtained via a Q-NET.

Anisotropy is controlled via the distribution of the W_1. Just as the location $\mathbf{x}_c$ of the middle of the step (where $\sigma(\mathbf{x}_c)=0.5$) is given by $\mathbf{x}_c=-\mathbf{b}_1/w_1$ in 1D, the orientation of the i^{th} 2D sigmoid is given by $W_1^{i,2}/W_1^{i,1}$. Randomness in W_1 can be guided to produce noise functions with different distributions of sigmoid-orientations as shown in the top row of Fig. 10 ($k = 25K$). The insets visualize the distribution of orientations.

Neural noise patterns scale well to multiple dimensions. The submitted video shows a scene containing a time-varying participating medium with heterogeneous density (3D + 1D). We modeled the 3D density in each frame as a slice (in time) of a 4D noise function with $k = 10K$ neurons. We rendered images using standard MC path tracing implemented in PBRT [PJH16]. Q-NETs enable analytical integration of optical depth along rays (Sec. 5.3).

5.3. Estimating optical depth

We represent a scalar density field using the neural proxy and integrate the density along rays by first performing appropriate transformations (translation to the ray origin and rotation to align one of the dimensions with the ray direction) and slicing. The natural logarithm of the ratio of incident to transmitted radiant power through a material, or *optical depth* τ, often needs to be estimated for rendering and volume visualization applications. The optical depth between two points is usually calculated via integration of $\alpha(s)$, the attenuation coefficient at a distance of s from the first point, over the line segment between the points. The integrals are usually estimated via sampling or ray-marching for heterogeneous media.

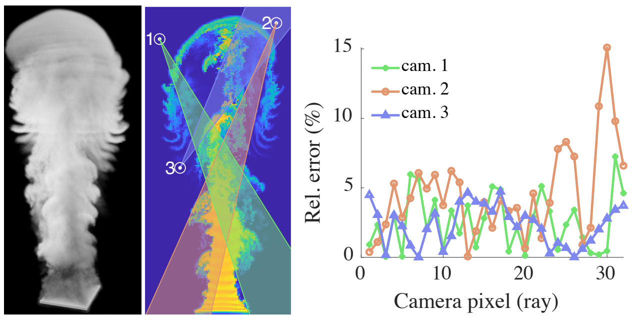


Figure 11: *We fit a proxy to density data from a smoke simulation in Blender and calculate optical depth along rays using a Q-NET. A frame from the simulation is shown rendered using Blender Cycles (left). For the central slice, we use a value proportional to the density as the attenuation coefficient and integrate it along 32 rays each for three virtual camera poses (middle). The plot (right) shows percentage relative error for the rays.*

We performed a smoke simulation using Blender, exported the $101 \times 100 \times 223$ spatially-varying density to a VDB file and trained $f_{\mathbf{W}}$ within the support volume (simulation domain). Given a ray specified by its origin and direction, we derive the transformation that aligns the X axis of $f_{\mathbf{W}}$ with the ray. Then, we slice $f_{\mathbf{W}}$ with $y = 0, z = 0$ and integrate along the remaining dimension to obtain the optical depth along the ray within the volume. Fig. 11.a visualizes a frame of the simulation rendered using Blender cycles. Fig. 11.b visualizes a vertical slice of the simulation along with three virtual 2D cameras. For each of the 32 pixels per camera, we estimated optical depth (until the ray leaves the volume) and plotted the relative errors (as percentages) in Fig. 11.c. The plots are representative of our experiments: 90% of the estimates were below 5% relative error and about 98% of the estimates are below 10%.

5.4. Visualizing flux through rectangular volumes

The flux J_s of a vector field $\vec{F}(\mathbf{x})$ through a cuboidal volume (voxel) of side s is usually calculated as a surface integral of $\vec{F}$ over the surface of the voxel. According to the divergence theorem,

$$J_s = \int_{V_s} \nabla.\vec{F}(\mathbf{x})\, d\mathbf{x} \tag{10}$$

where V_s is the volume of the voxel and $\nabla.\vec{F}(\mathbf{x}) \equiv \sum \partial\vec{F}/\partial\mathbf{x}^i$ is the divergence of $\vec{F}$. Thus, if $f_{\mathbf{W}}$ is trained to represent divergence then an approximation to the above integral can be calculated in closed-form for cuboidal neighborhoods. Fig. 12 shows this calculation applied to a medical dataset of deformation observed in a human lung via 4D CT scans [VSC*07]. The dataset provides a 3D deformation field during breathing, which we use as $\vec{F}$. The divergence of $\vec{F}$ then corresponds to the trace of the local strain tensor which is called *dilatation* (or dilation). The streamtube plots in Fig. 12b. and c. visualize local dilatation between two 3D frames (inspiration and expiration) along streamlines in $\vec{F}$. In Fig. 12d. we color streamtubes by the local flux within $5 \times 5 \times 5$-voxel neighborhoods. In this context, flux corresponds to average expansion (blue) or compression (red) of local neighborhoods during expiration. This method could also be useful to represent, interpolate and visualize flux in Lagrangian fluid simulations.

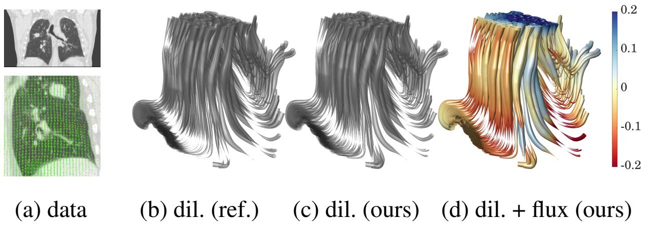


Figure 12: *(a) Images showing the deformation field in a human lung during respiration [VSC*07]. (b) Streamtubes for deformation between two frames of the dataset. The thickness of tubes corresponds to local dilatation. (c) Approximation dilatation obtained using a neural proxy. (d) Visualizing local flux (color) calculated by integrating the proxy.*

6. Discussion and future work

Domain and range transformation. In practice, we normalize the domain to $[-1,1]^d$ and the range to $[-1,1]$ via input and output (maxminmap) operators. The formula is corrected accordingly. i.e. In 1D, if $x \in [x_a, x_b]$, $f_{\mathbf{W}}(x) \in [z_a, z_b]$ and the normalized function is $\hat{f}_{\mathbf{W}}(\hat{x})$, then

$$f_{\mathbf{W}}(x) = \frac{z_b - z_a}{2}\left(\hat{f}_{\mathbf{W}}\left(\frac{x_b - x_a}{2}(\hat{x}+1) + x_a\right) + 1\right) + z_a.$$

Estimates in the normalized space $\hat{\mu}_{d,k}$ can be transformed to estimate $\mu_{d,k}$ in the required domain via variable substitution as $\mu_{d,k} =$

$\Omega \,.\, [(z_b - z_a)(\hat{\mu}_{d,k}/2^d + 1)/2 + z_a]$, where $\Omega = \prod_{j=1}^{d}(\mathbf{x}_b^j - \mathbf{x}_a^j)$ is the volume of the domain. We used this in all our empirical tests.

Loss functions and training. We did not notice significant differences in the convergence rate for different loss functions such as `mse`, `mae` and `cross-entropy`. We found that optimizing using the Levenberg-Marquardt method and Bayesian Regularization perform better than conjugate gradient based methods particularly for discontinuous f. We used the latter in our experiments because they are suited to training on GPUs.

Sampling. Q-NETs yield a consistent estimator regardless of the sampling distribution provided it is non-zero everywhere that f is non-zero. Low-fidelity reconstruction due to poor sampling could indeed increase variance. Variance is defined across estimates $\mu(\mathbf{w})$ from samples $\{\mathbf{x}_n\}$ due to stochasticity in the initialization or optimization. The error bars in our plots were generated using a fixed set of samples across repetitions for Q-NETs while different sets were used for MC and QMC. Although this is to our disadvantage, it highlights a benefit of the proxy which is to reduce unnecessary evaluations of potentially costly integrands. Low-discrepancy sampling appears to perform better than random sampling for training. The QMC methods that we tested (Halton with and without scrambling and Sobol) showed variable benefits across different test integrands but exhibited grossly similar convergence trends.

Gaussian Processes. Despite their success as surrogates, GPs cope poorly with discontinuities and scale slowly in N. We compared errors of Q-NETs with Bayesian Quadrature (BQ) using the EmuKit library [PPM*19] on a step function in 1D (100 repetitions). BQ performed $0.25\times$ better (lower error) at 16 samples but $3\times$ worse at 512. Q-NETs are faster than BQ by factors of $2\times$ (16 samples) and $400\times$ (512 samples) per rep. Discontinuities in higher dimensional functions pose a greater challenge to GPs than to neural proxies.

Limitation. When integrating discontinuous functions, excessively large k is counterproductive due to ringing artifacts caused due to overfitting. Ringing also occurs at discontinuities. Figure 13 shows that increasing N effectively reduces the variance of the proxy (± 2 standard deviations over 1000 different proxies is shown as a red shaded region) but does not completely eliminate this effect at discontinuities. The spread of the histograms of integral estimates reflect the variance of the estimators. Indeed, increasing the sampling rate is impractical in higher dimensions. Fortunately, the oscillatory nature of these artifacts poses fewer problems for integration bias than it does for reconstruction error or for integration variance. This well-known drawback is typically addressed by bounding width and adding layers [FWG*18]. Further work is required to extend the idea in this paper to multiple layers.

7. Conclusion and future work

We present a practical mechanism to calculate exact integrals of functions represented by shallow sigmoidal neural networks. Our family of fixed networks operates on the parameters of trained proxies to calculate marginals and integrals. In this paper we focused on quantitative assessment of the effectiveness of the proxy for functions with discontinuities and demonstrated its effectiveness for a variety of problems. In the future, we hope to extend the idea to multiple layers and hone this work to specific applications such as volume rendering and light transport simulation. We also foresee exciting possibilities such as rapid evaluation using implementation of the fixed-weight Q-NET family directly in hardware.

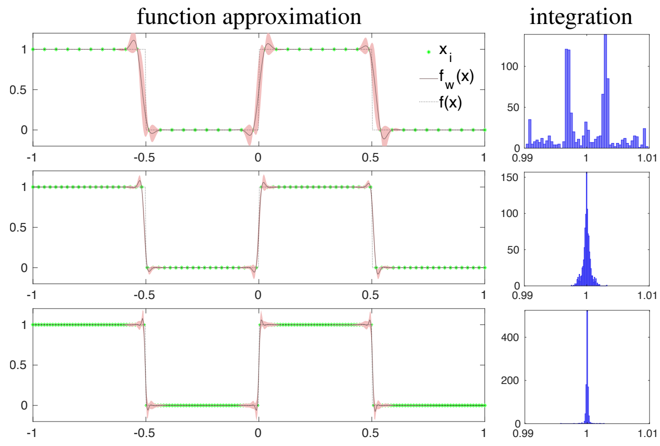


Figure 13: *Discontinuities pose a problem for the proxy (left) and its integral estimates (right). As the number of samples is increased from 40 (top) to 80 (middle) and 160 (bottom) samples with $k = 20$ neurons, the variance in the 'ringing' artifacts are reduced. The histogram of estimates obtained using Q-NET for each proxy (rows) is shown on the right. The reference is $I = 1.0$ for this example.*

Acknowledgements

The author was funded by a Royal Society University Research Fellowship. The 3D model of the bull used in Fig. 9 was submitted to Free3D.com by `printable_models`.

Appendix

The approximation error of shallow feedforward networks [Bar93, SCC15] is bounded by $||f(\mathbf{x}) - f_\mathbf{w}(\mathbf{x})||_2^2 < \varepsilon$ where $\varepsilon = \tilde{h}/k^{2c/d}$. $\tilde{h}$ is the first moment of the Fourier spectrum of f which is c times differentiable. Writing $f_\Delta(\mathbf{x}) \equiv f(\mathbf{x}) - f_\mathbf{w}(\mathbf{x})$,

$$\begin{aligned}
||f(\mathbf{x}) - f_\mathbf{w}(\mathbf{x})||_2^2 &= \int_{\mathcal{D}} f_\Delta^2(\mathbf{x})\,\mathrm{d}\mathbf{x} \\
&= \int_{\mathcal{D}} f_\Delta^2(x)\,\mathrm{d}\mathbf{x} - \left(\int_{\mathcal{D}} f_\Delta(\mathbf{x})\,\mathrm{d}\mathbf{x}\right)^2 + \left(\int_{\mathcal{D}} f_\Delta(\mathbf{x})\,\mathrm{d}\mathbf{x}\right)^2 \\
&= \mathrm{V}[f_\Delta(x)] + \left(\int_{\mathcal{D}} (f(\mathbf{x}) - f_\mathbf{w}(\mathbf{x}))\,\mathrm{d}\mathbf{x}\right)^2 \\
&= \mathrm{V}[f_\Delta(x)] + (I-\mu)^2
\end{aligned}$$

where V[.] is the variance operator. Substituting this into the bounds for approximation error we have $(I-\mu)^2 < \varepsilon - V[f(\mathbf{x}) - f_\mathbf{w}(\mathbf{x})]$.

References

[Ace18] Acerbi L.: Variational bayesian monte carlo. In *Advances in Neural Information Processing Systems 31* (2018), Bengio S., Wallach H. M., Larochelle H., Grauman K., Cesa-Bianchi N., Garnett R., (Eds.), pp. 8223–8233. 2

[AKJ08] An S. S., Kim T., James D. L.: Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. Graph. 27*, 5 (Dec. 2008). URL: https://doi.org/10.1145/1409060.1409118, doi:10.1145/1409060.1409118. 2

[Bar93] Barron A. R.: Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory 39*, 3 (1993), 930–945. 9

[BLD20] Bangaru S. P., Li T.-M., Durand F.: Unbiased warped-area sampling for differentiable rendering. *ACM Trans. Graph. 39*, 6 (Nov. 2020). URL: https://doi.org/10.1145/3414685.3417833, doi:10.1145/3414685.3417833. 6

[BOGO15] Briol F.-X., Oates C., Girolami M., Osborne M. A.: Frank-wolfe bayesian quadrature: Probabilistic integration with theoretical guarantees. In *Advances in Neural Information Processing Systems 28*, Cortes C., Lawrence N. D., Lee D. D., Sugiyama M., Garnett R., (Eds.). 2015, pp. 1162–1170. 2

[BP11] Brass H., Petras K.: *Quadrature Theory: The Theory of Numerical Integration on a Compact Interval*. Mathematical surveys and monographs. American Mathematical Society, 2011. 2

[BS19] Bargteil A. W., Shinar T.: An introduction to physics-based animation. In *ACM SIGGRAPH 2019 Courses* (2019), SIGGRAPH '19, Association for Computing Machinery. URL: https://doi.org/10.1145/3305366.3328050, doi:10.1145/3305366.3328050. 2

[BSS*13] Belcour L., Soler C., Subr K., Holzschuch N., Durand F.: 5d covariance tracing for efficient defocus and motion blur. *ACM Trans. Graph. 32*, 3 (July 2013). URL: https://doi.org/10.1145/2487228.2487239, doi:10.1145/2487228.2487239. 2

[CJ16] Christensen P. H., Jarosz W.: The path to path-traced movies. *Foundations and Trends in Computer Graphics and Vision 10*, 2 (Oct. 2016), 103–175. doi:10/gfjwjc. 2

[CNN20] Chen Z., Nobuhara S., Nishino K.: Invertible neural brdf for object inverse rendering. In *Computer Vision – ECCV 2020* (2020), Vedaldi A., Bischof H., Brox T., Frahm J.-M., (Eds.). 6

[Cra06] Crandall R. E.: Note on fast polylogarithm computation, 2006. https://www.reed.edu/physics/faculty/crandall/papers/Polylog.pdf. 3

[Cyb89] Cybenko G.: Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*, 4 (1989), 303–314. 1

[DHS*05] Durand F., Holzschuch N., Soler C., Chan E., Sillion F. X.: A frequency analysis of light transport. In *ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), SIGGRAPH '05, Association for Computing Machinery, p. 1115–1126. URL: https://doi.org/10.1145/1186822.1073320, doi:10.1145/1186822.1073320. 2

[Dur11] Durand F.: A frequency analysis of monte-carlo and other numerical integration schemes. *MIT-CSAIL-TR-2011-052* (2011). 1, 4

[Eul68] Euler L.: *Institutionum calculi integralis volumen primum*, vol. 1. 1768. English translation by Ian Bruce: http://www.17centurymaths.com/contents/integralcalculusvol1.htm. URL: https://archive.org/details/leonhardieuleri02eulegoog. 3

[FHH*19] Fascione L., Hanika J., Heckenberg D., Kulla C., Droske M., Schwarzhaupt J.: Path tracing in production: Part 1: Modern path tracing. In *ACM SIGGRAPH 2019 Courses* (2019), SIGGRAPH '19. URL: https://doi.org/10.1145/3305366.3328079, doi:10.1145/3305366.3328079. 2

[FWG*18] Fan F., Wang D., Guo H., Zhu Q., Yan P., Wang G., Yu H.: Slim, sparse, and shortcut networks. arXiv:1811.09003. 9

[FWKH17] Fong J., Wrenninge M., Kulla C., Habel R.: Production volume rendering. In *ACM SIGGRAPH 2017 Courses* (2017), SIGGRAPH '17. 2

[GJS19] Grohs P., Jentzen A., Salimova D.: Deep neural network approximations for monte carlo algorithms. arXiv:1908.10828. 2

[GLLD12] Galerne B., Lagae A., Lefebvre S., Drettakis G.: Gabor noise by example. *ACM Trans. Graph. 31*, 4 (July 2012). URL: https://doi.org/10.1145/2185520.2185569, doi:10.1145/2185520.2185569. 7

[GOG*14] Gunter T., Osborne M. A., Garnett R., Hennig P., Roberts S. J.: Sampling for inference in probabilistic models with fast bayesian quadrature. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (2014), NIPS'14, p. 2789–2797. 2

[GR03] Ghahramani Z., Rasmussen C. E.: Bayesian monte carlo. In *Advances in Neural Information Processing Systems 15*, Becker S., Thrun S., Obermayer K., (Eds.). MIT Press, 2003, pp. 505–512. 2

[GT96] Guenter B., Tumblin J.: Quadrature prefiltering for high quality antialiasing. *ACM Trans. Graph. 15*, 4 (Oct. 1996), 332–353. URL: https://doi.org/10.1145/234535.234540, doi:10.1145/234535.234540. 2

[HJW*08] Hachisuka T., Jarosz W., Weistroffer R. P., Dale K., Humphreys G., Zwicker M., Jensen H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 1–10. URL: https://doi.org/10.1145/1360612.1360632, doi:10.1145/1360612.1360632. 1

[Hor91] Hornik K.: Approximation capabilities of multilayer feedforward networks. *Neural networks 4*, 2 (1991), 251–257. 1

[JLSJ11] Johnson J. M., Lacewell D., Selle A., Jarosz W.: Gaussian quadrature for photon beams in tangled. In *ACM SIGGRAPH 2011 Talks* (2011). 2

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1986), SIGGRAPH '86, Association for Computing Machinery, p. 143–150. URL: https://doi.org/10.1145/15922.15902, doi:10.1145/15922.15902. 2

[KB20] Karaletsos T., Bui T. D.: Hierarchical gaussian process priors for bayesian neural network weights, 2020. arXiv:2002.04033. 2

[KDBB17] Koschier D., Deul C., Brand M., Bender J.: An hp-adaptive discretization algorithm for signed distance field generation. *IEEE Transactions on Visualization and Computer Graphics 23*, 10 (2017), 2208–2221. doi:10.1109/TVCG.2017.2730202. 1

[KH19] Kanagawa M., Hennig P.: Convergence guarantees for adaptive bayesian quadrature methods. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 6237–6248. 2

[KKN18] Keshavarzzadeh V., Kirby R. M., Narayan A. C.: Numerical integration in multiple dimensions with designed quadrature. *CoRR abs/1804.06501* (2018). URL: http://arxiv.org/abs/1804.06501, arXiv:1804.06501. 2

[KMA*15] Kettunen M., Manzi M., Aittala M., Lehtinen J., Durand F., Zwicker M.: Gradient-domain path tracing. *ACM Transactions on Graphics (TOG) 34*, 4 (2015). URL: https://doi.org/10.1145/2766997, doi:10.1145/2766997. 2

[KSP15] Kandasamy K., Schneider J., Póczos B.: Bayesian active learning for posterior estimation. In *Proceedings of the 24th International Conference on Artificial Intelligence* (2015), IJCAI'15, p. 3605–3611. 2

[KYT*06] Kharevych L., Yang W., Tong Y., Kanso E., Marsden J. E., Schröder P., Desbrun M.: Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM*

SIGGRAPH/Eurographics symposium on Computer animation (2006), pp. 43–51. 2

[LD11] LAGAE A., DRETTAKIS G.: Filtering solid gabor noise. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2011) 30*, 4 (July 2011), 51:1–51:6. doi:10.1145/1964921.1964946. 7

[Lew81] LEWIN L.: *Polylogarithms and associated functions*. North Holland, 1981. 3

[LHK*20] LAINE S., HELLSTEN J., KARRAS T., SEOL Y., LEHTINEN J., AILA T.: Modular primitives for high-performance differentiable rendering. *ACM Trans. Graph. 39*, 6 (Nov. 2020). URL: https://doi.org/10.1145/3414685.3417861, doi:10.1145/3414685.3417861. 6

[LIA20] LLOYD S., IRANI R., AHMADI M.: Using neural networks for fast numerical integration and optimization. *IEEE Access 8* (2020), 84519–84531. doi:10.1109/ACCESS.2020.2991966. 1, 2, 3

[LKL*13] LEHTINEN J., KARRAS T., LAINE S., AITTALA M., DURAND F., AILA T.: Gradient-domain metropolis light transport. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 1–12. 2

[LMW20] LINDELL D. B., MARTEL J. N. P., WETZSTEIN G.: Autoint: Automatic integration for fast neural volume rendering, 2020. arXiv:2012.01714. 2

[LPW*17] LU Z., PU H., WANG F., HU Z., WANG L.: The expressive power of neural networks: A view from the width. arXiv:1709.02540. 1

[MRKN20] MÜLLER T., ROUSSELLE F., KELLER A., NOVÁK J.: Neural control variates. *ACM Transactions on Graphics (TOG) 39*, 6 (2020), 1–19. doi:10.1145/3414685.3417804. 2

[MU49] METROPOLIS N., ULAM S.: The monte carlo method. *Journal of the American Statistical Association 44*, 247 (1949), 335–341. 2

[NDVZJ19] NIMIER-DAVID M., VICINI D., ZELTNER T., JAKOB W.: Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia) 38*, 6 (Dec. 2019). doi:10.1145/3355089.3356498. 6, 7

[Nie78] NIEDERREITER H.: Quasi-monte carlo methods and pseudo-random numbers. *Bulletin of the American Mathematical Society* (1978). 2

[Nie92] NIEDERREITER H.: *Random Number Generation and Quasi-Monte Carlo Methods*. 1992. doi:10.1137/1.9781611970081.fm. 2

[ODR09] OVERBECK R. S., DONNER C., RAMAMOORTHI R.: Adaptive wavelet rendering. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 1–12. URL: https://doi.org/10.1145/1618452.1618486, doi:10.1145/1618452.1618486. 2

[OGG*12] OSBORNE M., GARNETT R., GHAHRAMANI Z., DUVENAUD D. K., ROBERTS S. J., RASMUSSEN C. E.: Active learning of model evidence using bayesian quadrature. In *Advances in Neural Information Processing Systems 25*. 2012, pp. 46–54. 2

[Owe13] OWEN A. B.: *Monte Carlo theory, methods and examples*. 2013. Accessed on January 11 2021. URL: https://statweb.stanford.edu/~owen/mc/. 1, 2, 5

[PDW20] PETROSYAN A., DEREVENTSOV A., WEBSTER C. G.: Neural network integral representations with the relu activation function. In *Mathematical and Scientific Machine Learning* (2020), PMLR, pp. 128–143. 2

[Per02] PERLIN K.: Improving noise. *ACM Trans. Graph. 21*, 3 (July 2002), 681–682. URL: https://doi.org/10.1145/566654.566636, doi:10.1145/566654.566636. 7

[PH89] PERLIN K. H., HOFFERT E. M.: Hypertexture. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (1989), SIGGRAPH '89, p. 253–262. URL: https://doi.org/10.1145/74333.74359, doi:10.1145/74333.74359. 2, 7

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation (3rd ed.)*, 3rd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, Oct. 2016. 8

[PPM*19] PALEYES A., PULLIN M., MAHSERECI M., LAWRENCE N., GONZÁLEZ J.: *Emulation of physical processes with Emukit*. Second Workshop on Machine Learning and the Physical Sciences, NeurIPS, 2019. 9

[RAMN12] RAMAMOORTHI R., ANDERSON J., MEYER M., NOWROUZEZAHRAI D.: A theory of monte carlo visibility sampling. *ACM Trans. Graph. 31*, 5 (2012). URL: https://doi.org/10.1145/2231816.2231819, doi:10.1145/2231816.2231819. 1, 4

[SCC15] SHAHAM U., CLONINGER A., COIFMAN R. R.: Provable approximation properties for deep neural networks. arXiv:1509.07385. 1, 9

[SK13] SUBR K., KAUTZ J.: Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration. *ACM Trans. Graph. 32*, 4 (July 2013). URL: https://doi.org/10.1145/2461912.2462013, doi:10.1145/2461912.2462013. 1, 4

[SSW*16] SHAHRIARI B., SWERSKY K., WANG Z., ADAMS R. P., DE FREITAS N.: Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE 104*, 1 (2016), 148–175. 2

[Stu10] STUART A. M.: Inverse problems: A bayesian perspective. *Acta Numerica 19* (2010), 451–559. doi:10.1017/S0962492910000061. 6

[Sub20] SUBR K.: Q-net: A network for low-dimensional integrals of neural proxies, 2020. arXiv:2006.14396v1. 2, 3

[TNVdVG19] TEICHERT G., NATARAJAN A., VAN DER VEN A., GARIKIPATI K.: Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions. *Computer Methods in Applied Mechanics and Engineering 353* (Aug 2019), 201–216. URL: http://dx.doi.org/10.1016/j.cma.2019.05.019, doi:10.1016/j.cma.2019.05.019. 2

[Vea98] VEACH E.: *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162. 2

[VSC*07] VANDEMEULEBROUCKE J., SARRUT D., CLARYSSE P., ET AL.: The popi-model, a point-validated pixel-based breathing thorax model. *Proc of ICCR, 2007 2* (2007), 195–199. 8

[WL18] WANG H., LI J.: Adaptive gaussian process approximation for bayesian inference with expensive likelihood functions. *Neural Computation 30*, 11 (2018), 3072–3094. PMID: 30216145. arXiv:https://doi.org/10.1162/neco_a_01127. 2

[WLF*20] WANG X., LI M., FANG Y., ZHANG X., GAO M., TANG M., KAUFMAN D. M., JIANG C.: Hierarchical optimization time integration for cfl-rate mpm stepping. *ACM Trans. Graph. 39*, 3 (2020). URL: https://doi.org/10.1145/3386760, doi:10.1145/3386760. 2

[YDW13] YAN L., DI J., WANG K.: Spline basis neural network algorithm for numerical integration. *International Journal of Mathematical and Computational Sciences 7*, 3 (2013), 458 – 461. URL: https://publications.waset.org/vol/75. 2

[ZWZ*19] ZHANG C., WU L., ZHENG C., GKIOULEKAS I., RAMAMOORTHI R., ZHAO S.: A differential theory of radiative transfer. *ACM Trans. Graph. 38*, 6 (Nov. 2019). URL: https://doi.org/10.1145/3355089.3356522, doi:10.1145/3355089.3356522. 6

[ZZYNH06] ZHE-ZHAO Z., YAO-NAN W., HUI W.: Numerical integration based on a neural network algorithm. *Computing in Science and Engineering 8* (08 2006), 42–48. doi:10.1109/MCSE.2006.73. 2